APPLE PEEK,POKE 응용

한 수 찬 著



數學社

APPLE PEEK,POKE 응용

한 수 찬 著



教學社

책을 내면서

지금 우리나라에 무서운 속도로 번지고 있는 퍼스널 컴퓨터의 열기는 날이 갈수록 더 심해지고 있다는 느낌이다. 그 동안 여러 기종의 퍼스널 컴퓨터가 나왔지만 여전히 왕좌를 지키고 있는 것은 역시 Apple Ⅱ로서 그 광범위한 사용자의 층은 쉽사리 변화될 것 같지는 않다.

그런데 Apple Ⅱ의 사용자 중 **대부분의 사람들이 BASIC을 어느 정도 이해하고** 난 후에도, **PEEK, POKE의 개념을 잡지못하겠다고 푸념**하고 있다.

왜 그럴까?

대부분의 책들이 PEEK, POKE에 대해서는 자세한 언급이 없을 뿐더러, 몇 가지 특수한 기능만 나열해 놓고 그 개념과 응용방법에 대해서 거의 설명이 안되어 있기 때문이다. 또한 **PEEK' POKE는 각각의 컴퓨터의 구조에 따라서 매우 특수하고 오묘한 기능을 수행하므로** 이것을 잘 알아 둔다면 여러분은 여러분의 **컴퓨터 구석구석까지를 최대한 활용**할 수 있게 되는 것이다. 이 책은 Apple 컴퓨터를 보다 효율적으로 사용하게 한다는 취지아래 집필되었다.

사실 POKE문에 의한 **TEXT 윈도우의 사용법만 제대로 알아**도 BASIC 프로그램의 수준은 서너 계단 껑충 뛰어 오르게 될 것이다. 또한 이 책에서는 **Apple** 컴퓨터에서 **실제로 응용 가능하며, 또 그 가치가 있는 것만을 엄선**하였으므로 불필요한 부분을

최소한으로 하려하였다.

그러나 특정 컴퓨터의 특정 부분인 관계로 자료가 거의 없었고 저자의 능력이 미치지 못함으로 해서 그 많은 컴퓨터 책에 또 하나의 불필요한 책을 더하지 않았는지 지금의 심정은 두렵기만 하다. 다만 Apple의 본산지인 미국에도 없는 종류의 책을 만들었다는 데에 조그마한 위안을 삼고 싶다.

이 책이 나오기까지 여러분의 도움이 있었다. 서울대학교 공과대학의 양 홍석 교수님, 지 철근 교수님, 황 희융 교수님, 한 송엽 교수님께 감사드린다. 특히 양 교수님의 격려 편지는 잊을 수 없으리라. 또한 이 책의 발간에 온갖 배려를 다해주신 교학사 楊澈愚 사장님과 컴퓨터과의 강 대철 과장님께 감사드리며, 자료를 협조해 주신 한도 엔지니어링 여러분, 그리고 프로그램의 출력에 애써주신 양 정미씨에게 고마운 마음 전해 드린다.

장마가 시작 된 것인지 창 밖에는 계속 비가 내리고 있다. 올 가을엔 뭐 좀 좋은 일 없을까.

1984. 하지 날

저자 씀

차 례

1부 PEEK, POKE, CALL의 개념

2부 PEEK, POKE, CALL의 응용

1 부

PEEK, POKE, CALL의 개념



1장 메모리란 무엇인가?

메모리란 무엇인가?

PEEK, POKE 등의 명령어를 이해하려면 우선 메모리에 관한 개념부터 갖추고 있어야 한다. 왜냐하면 이들 명령어는 특정한 메모리에 직접 관여하는 명령어이기 때문이다.

일반적으로 메모리는 기억장소라고 말할 수 있는데, 컴퓨터 내의 반도체 칩① 안에 아파트 방처럼 수많은 기억장소들이 준비되어 있다. 그 각각의 방 안에 정보를 집어 넣기도 하고, 또 이미 들어가 있는 정보를 끄집어 내기도 한다.

여기서 말하는 정보는, 프로그램 자체와 이 프로그램이 사용하게 되는 데이터들을 말하는데 **이 자료들은 메모리가 없이는 컴퓨터 내에 존재할 수가 없는 것**이다. 이 각각의 메모리에는 일련의

고유번호가 붙어있는데 이것을 "**메모리 어드레스**(memory address)"라고 한다. 아파트의 주소, 혹은 호수를 생각하면 될 것이다. 메모리 어드레스를 사용하여 특정한 위치의 메모리를 지정해 그곳에 정보를 담아두거나 그곳에 있던 정보를 끄집어 내오게 되는 것이다.

메모리 어드레스가 없다면 어떤 장소에 데이터를 넣어야 할지, 또 어느 장소의 데이터를 프로그램이 원하고 있는지 알 수 없게 된다.

메모리 칩은 크게 두 가지로 분류할 수 있는데 하나는 **RAM**(random aceess memory)이라 불리우고, 다른 하나는 **ROM**(read only memory)이라 불리운다. Apple의 뚜껑을 열어보면 키이보오드 있는 쪽으로 나란히 6개의 조금 큰 IC[3]가 붙어있는데 이것이 ROM이다.

ROM은 그곳에 정보를 기록해 넣을 수 없고 미리 기록된 정보만을 끄집어 낼 수(read) 있다. 따라서 ROM 안의 정보는 컴퓨터가 발매되어 나올 때 이미 공장에서 다 기록되어 나오는 것이며 사용자는 프로그램상에서 그 내용을 고칠 수 없다.

Apple의 ROM 안에는 사용자가 짠 BASIC 프로그램을, 컴퓨터가 수행할 수 있는 기계어로 바꾸는 번역 프로그램과 컴퓨터에 전체적인 통제를 하는 오퍼레이팅 시스템이라는 관리 프로그램이 들어가 있다.

RAM은 사용자가 정보를 기록해 넣을 수도 있고 넣었던 정보를 다시 읽어낼 수도 있는 메모리 장치로서 프로그램이나 데이터가 들어갈 장소이다.

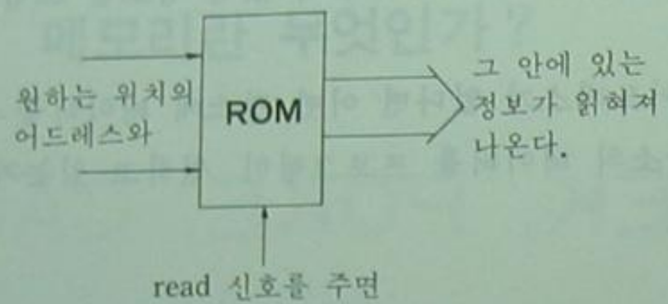


그림 1.1 ROM의 동작

Apple 기판의 한 가운데를 보면 가로로 8개씩 세로 3줄의 작은 IC칩을 발견할 수 있을 것이다.

Apple의 메모리 용량③은 2^{16}=64 K 바이트(byte: 8 bit로 이루어지는 데이터의 기본 단위)로서 이는 바로 각 메모리 번지수의 갯수와도 같다(K : 1024를 뜻함).

즉 메모리 어드레스는 0~(2^{16}−1) 번지로서 **십진수 0~65535까지의** 65536개의 기억장소를 가리키고 있다(※ 주의 : BASIC의 문장번호와는 관련이 없음).

Apple에서는 이 64 K의 메모리 중, 사용자가 사용할 수 있는 RAM 영역은 48 K이고, 4 K는 입출력용으로 쓰이며 나머지 12 K는 ROM으로써 BASIC 인터프리터④와 모니터⑤ ROM으로 쓰이고 있다.

자세한 메모리 구성은 부록을 참조하기 바란다.

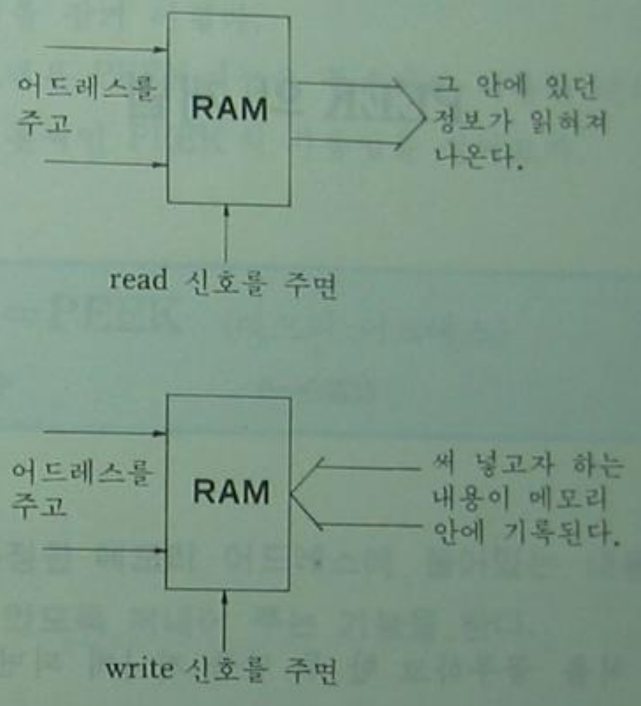

어드레스를
주고
RAM
그 안에 있던
정보가 읽혀져
나온다.
read 신호를 주면
어드레스를
주고
RAM
써 넣고자 하는
내용이 메모리
안에 기록된다.
write 신호를 주면

그림 1.2 RAM의 동작

2장 PEEK의 개념

Apple을 처음 공부하고 한 두 달쯤 지나게 되면 "BASIC이란 대충 이런 것이구나…"하는 생각과 조그만 자신감을 갖게 되면서도 대부분의 사람들이 "PEEK POKE는 아무리 봐도 잘 모르겠어 설명도 불충분하고…"하는 푸념을 늘어놓게 된다.

왜 그럴까. 그것은 대부분의 Apple 매뉴얼들이 PEEK, POKE의 개념은 설명하지 않고 무조건 특수한 기능을 하는 PEEK, POKE문만을 나열해 두었기 때문이다.

예를 들어 다음과 같은 BASIC 프로그램의 경우

```
10 X=PEEK (-16336)
20 GOTO 10
```

RUN을 시키면 '부~'하고 버저소리가 계속 울리게 되는데 어째서 그런지 이유를 잘 알 수가 없다.

이러한 PEEK의 사용을 처음으로 맞부딪히고 보면 실제의 PEEK의 개념을 잡기 어렵다.

위의 프로그램은 PEEK 기능을 특수하게 사용한 예이기 때문이다. 그러면 기본적인 PEEK의 사용법을 살펴보자.

X＝PEEK (메모리 어드레스)
↑ 변수 ↑ 0～65535

PEEK는 특정한 메모리 어드레스에 들어있는 내용을 BASIC 상에서 쓸 수 있도록 꺼내어 주는 기능을 한다.

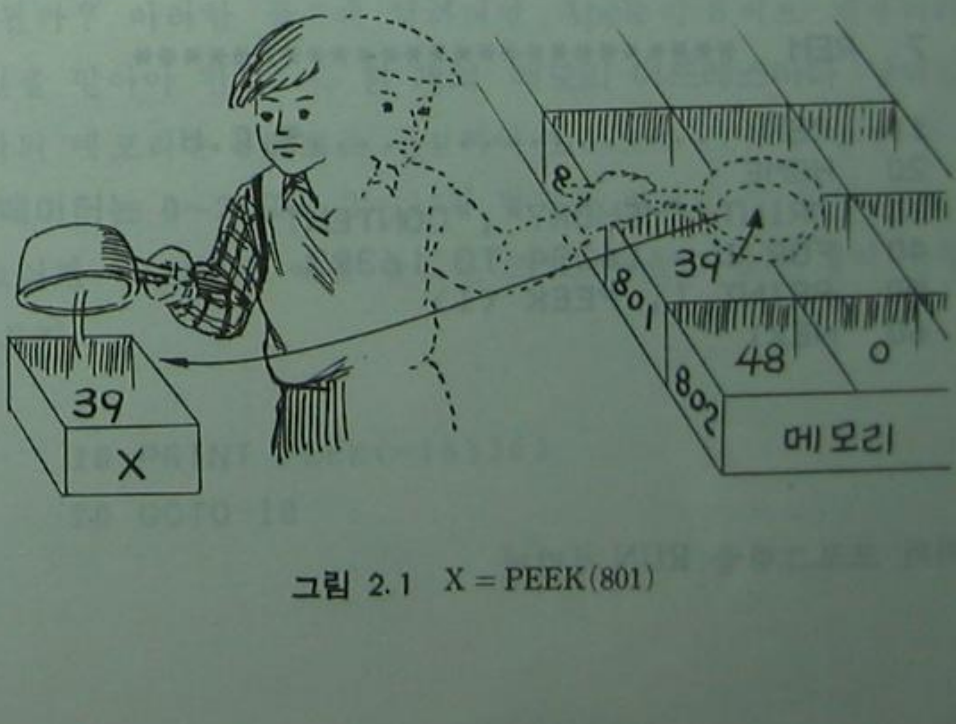


그림 2.1 X＝PEEK(801)

끄집어 낸 후에도 메모리에 있던 내용은 없어지지 않는다. 단지 복사를 해준 것 뿐이기 때문이다.

FORTRAN 같은 언어에서는 이같이 직접 메모리의 내용을 읽어내는 기능이 없다. 이러한 PEEK의 기능은 BASIC의 장점인 것이다. 메모리에 직접 작용하는 프로그램 명령어를 사용하게 되면 컴퓨터를 구석구석까지 사용할 수 있게 될 것이다.

예제 Apple의 메모리 번지 16384~16393까지의 내용을 메모리 어드레스와 함께 프린트하라.

```
]LIST

1  REM  PEEK1
5  REM  ***************************

6  REM  *  PEEK 16384--16393      *

7  REM  ***************************

10  REM  ...................S.C.H
20  HOME
30  PRINT "MEMORY","CONTENT"
40  FOR I = 16384 TO 16393
50  PRINT I, PEEK (I)
60  NEXT
```

위의 프로그램을 RUN 시키면

```
RUN
MEMORY          CONTENT
16384           223  (다른 값이 올 수도 있다)
16385           223
16386           223
16387           223
16388           223
16389           223
16390           223
16391           223
16392           255
16393           255
```

와 같은 결과가 찍힌다.

즉, 16384번지~16393번지의 메모리에는 223과 255라는 데이터가 들어가 있다는 뜻이다.

이러한 PEEK문에 의해 읽혀질 수 있는 데이터의 범위는 어떻게 될까? 이러한 물음에 답하려면 Apple이 8비트 컴퓨터라는 사실을 알아야 한다. 즉 **한 개의 메모리 어드레스마다** 달려있는 각각의 메모리는 **8비트**로 구성되어 있으므로 그곳에 담길 수 있는 **데이터는 0~255**$(0\sim2^8-1)$의 범위를 벗어날 수 없다.

그러면 맨 앞에서 예를 들었던 프로그램을 다음과 같이 변형시켜 보자.

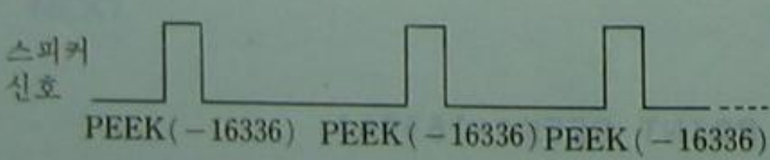
```
10 PRINT PEEK(-16336)
20 GOTO 10
```

RUN시키면 "부~"하는 소리와 함께 −16336번지의 값이 화면에 찍혀 나갈 것이다.

그런데 −16336번지의 −는 또 무엇인가. 앞의 설명으로는 메모리 어드레스 범위가 0~65535사이에 있어야 한다. 그러나 이 −는 Apple 내에서 데이터가 표시되는 방법 때문에 붙은 것이지 실제로 −의 어드레스는 없다. **이런 마이너스(−) 어드레스를 정상적인 어드레스로 바꾸어 주려면 그곳에 65536을 더해 주면** 된다. 즉 −16336번지는 −16336+65536=49200번지와 동일하다. PEEK(49200)으로 고쳐도 마찬가지 효과를 얻을 수 있다.

RUN되는 동안 화면에 찍혀 나가는 −16336번지의 값을 보니 일정한 값이 아니다. 255도 있고 0도 있고 160도 있고 여러 가지로 변해나가니 이해가 안간다.

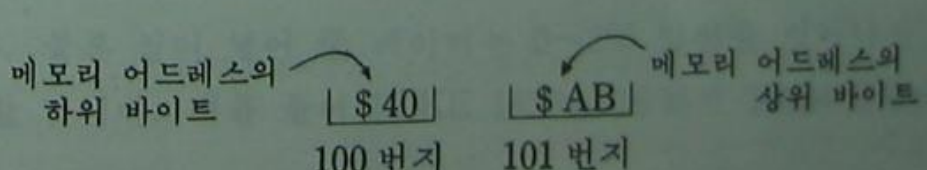
여기서 쓰인 PEEK의 기능은 −16336번지의 값을 끄집어 내는 것이 아니라 단지 PEEK(−16336)이라는 BASIC 명령을 수행함으로써 −16336번지, 즉 49200번지를 컴퓨터가 한 번 지정해 줄 수 있도록 하자는 것이다. 이 때 메모리 어드레스 49200번지가 지정될 때마다 Apple의 내장 스피커에 전기적 신호를 주도록 회로를 구성하여 놓으면 Apple의 스피커는 그 어드레스가 지정될 때마다 한 번씩 '틱'하고 울리게 될 것이다.



이것을 GOTO 루우프에 의해서 반복적으로 수행시키면 연속된 "부우~"하는 음이 들리게 되는 것이다. 이렇듯 **어떤 메모리 어드레스는 Apple의 독특한 하드웨어**(예 : 스피커, 키이보오드)에 **연결**되어 있기 때문에 그 어드레스를 지정해 주는 것만으로 특수한 동작을 하게 되며, 이 때의 PEEK된 값은 실제로 사용되는 값은 아니다. 또 사용할 수 있도록 RAM이나 ROM 메모리가 할당되어 있지도 않으며 **다만 사용자는 그 어드레스만을 이용할 뿐이다.** 그래서 앞의 예에서 프린트되는 PEEK 값이 불안정하게 변했던 것이다.

이렇게 Apple의 특수한 내부 하드웨어에 연결된 메모리는 49152~49279 번지에만 존재하고 있으므로 이 범위의 어드레스를 PEEK 문에 사용할 경우에는 그 값을 꺼내려는 것이 아님을 기억해 둔다면 앞으로 설명되는 PEEK나 POKE 명령어를 사용함에 큰 혼란이 없을 것이다. 이렇듯 **PEEK 문을 사용할 때에는 그 값을 꺼내어 보려는 경우와, 연결된 하드웨어를 동작시키기 위한 두 가지 경우**가 있다는 것을 알아두자.

또 Apple 내에서 연이어져 있는 두 장소를 16비트 어드레스를 담고 있는 곳으로 사용할 때가 많은데, 주로 상위 바이트가 뒤에 온다. 예를 들어 16진 어드레스 $AB40이 십진 100번지와 101번지에 저장되어 있고 두 개의 PEEK 문으로써 이것을 꺼내어 보고 싶다고 하자.



상위 바이트는 하위 바이트의 자리보다 256배의 자리수를 갖게 되므로

```
PRINT PEEK(100)+PEEK(101)*256
```

한다면 $AB40의 십진수값인 43840이 찍히게 될 것이다. 이러한 기법은 뒤에 나오는 응용문제들에서 많이 활용되므로 꼭 이해해 두도록 하자.

[참고] 4자리 16진수

$ [1] [2] [3] [4]

↑ ↑ ↑ ↑

$16^3=4096$ $16^2=256$ $16^1=16$ $16^0=1$

$=(4096\times \$1)+(256\times \$2)+(16\times \$3)+(1\times \$4)$

$=(\$12\times 256)+(\$34\times 1)$

상위 바이트 하위 바이트

($가 붙으면 16진수라는 뜻임)

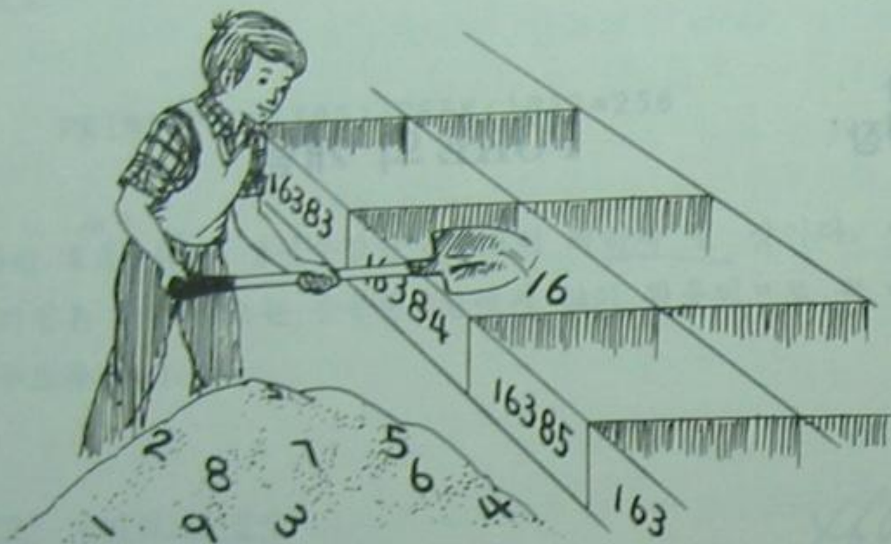
3장 POKE의 개념

POKE는 PEEK와는 대조가 되는 개념으로서, 사용자가 원하는 데이터를 어떤 메모리 장소에 강제적으로 넣어 주는 기능을 하는 BASIC 명령어이다. 물론 이 경우에도 특정한 어드레스를 십진수로 지정해 주어야 하는데 그 형태는,

> **POKE** 십진어드레스, 데이터

이다. 물론 집어 넣어 줄 데이터는 0~255 범위를 벗어나지 말아야 할 것이다. 예를 들어 POKE 16384, 16 하면 16384 번지에 16

이라는 데이터를 집어 넣게 된다.



그러면 다음 예제를 통해 POKE의 작용을 살펴보기로 하자.

예제 메모리 8001~8010 번지의 10개 내용을 PEEK 문을 사용하여 프린트해 보고, 다시 POKE 문을 사용하여 차례로 1, 2, 3, …, 10의 데이터를 8001~8010 번지에 넣으면서 POKE한 직후의 메모리 내용을 프린트 해 보아라.

```
]LIST

1  REM  P01
5  REM  *************************
6  REM  *  POKE 1 TO 10         *
7  REM  *************************
10  REM ....................S.C.H
20  FOR I = 8001 TO 8010
```

```
30  PRINT "ADDR:";I,"DATA="; PEEK (I)
40  NEXT
45  PRINT "AFTER POKING......................
    ......."
50  FOR I = 8001 TO 8010
60 K = K + 1: POKE I,K
70  PRINT "ADDR:";I,"DATA="; PEEK (I)
80  NEXT
```

K=K+1에 의해서 K는 1~10까지 변화하면서 POKE를 사용하여 그 값을 8001~8010번지에 넣어 주게 되므로 이 프로그램을 RUN시키면 다음과 같이 될 것이다.

```
]RUN
ADDR:8001       DATA=0
ADDR:8002       DATA=0
ADDR:8003       DATA=0
ADDR:8004       DATA=0
ADDR:8005       DATA=0
ADDR:8006       DATA=0
ADDR:8007       DATA=0
ADDR:8008       DATA=8
ADDR:8009       DATA=0
ADDR:8010       DATA=8
AFTER POKING.............................
ADDR:8001       DATA=1
ADDR:8002       DATA=2
ADDR:8003       DATA=3
ADDR:8004       DATA=4
ADDR:8005       DATA=5
ADDR:8006       DATA=6
ADDR:8007       DATA=7
ADDR:8008       DATA=8
ADDR:8009       DATA=9
ADDR:8010       DATA=10
```

이번에는 좀더 시각적으로 느낄 수 있는 예제를 통하여 POKE의 개념을 살펴보도록 하자.

예제 POKE 문을 사용하여 HGR 2 화면상에 흰 가로줄을 그어 나가면서 화면 전체를 칠해 보아라.

풀이 Apple의 HGR 2 화면은 메모리 16384~24575 번지에 할당되어 있으므로 우리는 이들 메모리에 어떤 데이터를 넣어주기만 하면 그대로 화면상에 표시될 것이다.

1 비트의 데이터는 HGR 화면상에 한 점으로 나타나게 될 것이므로 우리는 여기서 전 8 비트에 1을 넣어 주는

$$11111111_2 = \$FF = 255$$

의 데이터를 넣도록 하자.

사실상 HGR 화면상에 표시되는 한 메모리의 데이터는 7 비트뿐이고 나머지 한 비트(b_7)는 컬러를 지정하는데 쓰이므로 255 대신 127을 넣어도 같은 모양의 선이 그어질 것이다.

이 프로그램을 RUN 시켜 보면 HGR 2의 메모리 영역이 위에서부터 차례로 연결된 것이 아니고 화면을 3등분 해 나가면서 이어지고 있다는 것을 시각적으로 확인할 수 있다.

```
]LIST

1  REM  PO2
5  REM  ***************************
6  REM  *  POKE INTO HGR2         *
7  REM  ***************************
10  REM ......................S.C.H
20  HGR2
30  FOR I = 16384 TO 24575
40  POKE I,255
50  NEXT
```

이번에는 40번 문장의 데이터 255를 16으로 바꾸어보자. 255와 15는 각각 다음과 같은 비트 패턴을 갖는다.

1	1	1	1	1	1	1	1

255 — b_7은 화면에 표시 안됨

0	0	0	0	1	1	1	1

15 — b_7은 화면에 표시 안됨

그러므로 255가 짤막한 선분이었다면 16은 그 길이의 약 1/2에 해당하는 선분으로 나타나므로 전체적으로 화면에는 점선처럼 보이게 된다.

실제로 RUN시켜서 확인해 보자.

```
(40 POKE I, 16)
```

HGR 화면을 사용하여서 같은 실험을 해 볼 수 있다. 이 때 메모리 어드레스는 8192~16383으로 바뀌어야 할 것이다(부록의 'Apple 메모리 구성' 참조).

이상은 데이터를 집어 넣는다는 기본 개념을 그대로 적용한 정통파적인 POKE의 사용법이었다. 그러나 **POKE에도 PEEK에서처럼 그 어드레스만을 사용할 목적으로 쓰는 경우**가 있다. 이런 경우에는 POKE하는 어드레스만이 중요할 뿐, 집어 넣는 데이터는 0~255 중 어느 것이라도 괜찮으나 관례상 0을 쓰는 것이 일반적이다.

예를 들어

```
POKE -16304, 0 이나
POKE -16304, 200 은
```

모두 화면을 그래픽 모우드로 변환시키는 스위치 구실을 하게 된다.

4장 CALL의 개념

CALL은 BASIC에서 기계어 루우틴(기계어로 된 하나의 프로그램)을 **부를 때 사용되는 명령**이다. 이 경우에도 기계어 루우틴이 시작되는 어드레스를 알아서, 그 번지를 CALL해 주면 프로그램의 수행은 BASIC에서 곧장 그 기계어 루우틴으로 옮겨져서 기계어 프로그램을 진행시키게 된다.

CALL 십진 어드레스

사용자가 직접 기계어 서브루우틴을 작성한 경우에는 그 루우

틴이 시작되는 어드레스를 CALL하기만 하면 서브루우틴은 자동으로 수행되며, 서브루우틴의 끝에 RTS(서브루우틴에서의 복귀 명령)를 써 주면 다시 BASIC으로 돌아올 수 있다.

또 사용자가 작성한 서브루우틴이 아니고 이미 Apple의 ROM 속에 들어 있는 기계어 루우틴을 이용하고자 할 때에도 CALL 명령어가 무척 많이 사용되는데 그 대표적인 예가 CALL −151 이다.

−151 번지에는 모니터 모우드(*)로 들어가는 기계어 프로그램이 시작되므로 BASIC에서 이 번지를 CALL해 주기만 하면 모니터(*) 모우드로 들어가게 된다.

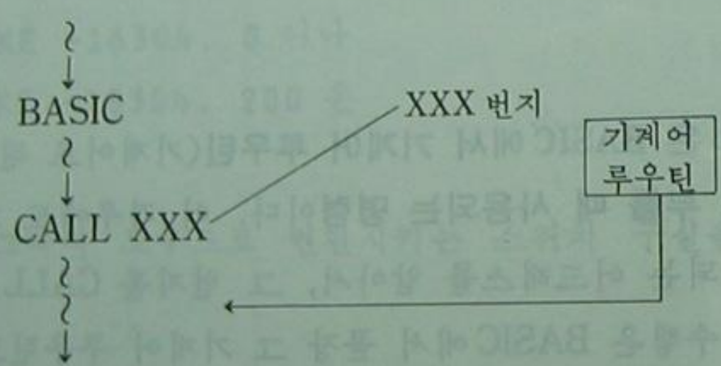


그림 4.1 CALL의 수행 과정

이러한 특수한 루우틴들을 사용하기 위해서는 각각의 어드레스를 훤히 알아야 하고 이것은 PEEK, POKE에서도 마찬가지이다.

이 책의 2부에서는 이러한 어드레스를 철저히 찾아내어서 그 활용 방법을 덧붙여 주었다. 이들을 최대로 활용한다면 한 차원 높은 소프트웨어를 만들 수 있게 될 것이다.

2 부

PEEK, POKE, CALL의 응용



응용 1

〈RUN 플랙〉 $D6

```
POKE 214, 255
```

모든 명령어가 프로그램을 RUN 시키게 한다.

【프로그램 리스트를 못보게 하자】

214번지는 통상 127 이하의 값을 가지고 있는데 이것을 POKE 명령으로 127보다 크게 만들면 LIST, RUN, LOAD 등의 **어떠한 명령어도 프로그램을 다시 RUN 시키고 만다.** 그러므로 당신이 애써 짜 놓은 프로그램을 남이 못 보도록 만들려면 위의 POKE 명령을 프로그램의 맨 앞에 넣어주면 된다. 단 이것은 CATALOG나 PR#6 등의 DOS 명령에는 듣지 않는다.

만일 프로그램의 어떤 부분부터는 다시 모든 명령이 정상적으로 동작하게 하려면 그 부분에 가서

```
POKE 214, 0
```

하면 된다.

예

```
]LOAD EX1
]LIST

3  REM  EX1
4  REM  ************
5  REM  * RUN FLAG *
6  REM  ************
7  REM ..................S.C.H
```

```
10  POKE 214,255
20  HOME : VTAB 5: PRINT "RUN..."
30  FOR I = 1 TO 10
40  PRINT I,I * 10,I * 100
50  NEXT I
60  FOR P = 1 TO 1000: NEXT P
70  PRINT : PRINT "-- YOU CAN'T SEE
    MY PROGRAM LIST ! --"
```

위 프로그램의 20번에서 70번까지를 실제의 프로그램이라 할 때 이 프로그램을 일단 한번 수행시키고 나면 아무리 LIST를 보려고 해도 불가능하게 될 것이다.

이 프로그램으로 디스켓을 이니셜라이즈(initialize, 初期化)시키면 더욱 좋은 효과를 내게 된다.

응용 2

〈ONERR 플랙〉 $D8

```
POKE 216, 0
```

ONERR GOTO 기능을 취소시킨다.

【ONERR GOTO 문의 영향권에서 벗어나려면】

ONERR GOTO XXX 라는 명령이 일단 수행이 되기만 하면 다음번 ONERR GOTO XX 라는 문장을 만나기 전까지는, 에러가 생기면 무조건 XXX 번 문장으로 점프해 가게 된다. 그러나 프로그램을 작성하다 보면 어느 순간에 가서는 ONERR GOSUB 기능을 철회할 필요가 생긴다. 이 때 사용하는 것이 바로 216 번지이다.

이 번지는 원래에는 127 이하의 값을 갖고 있다가 ONERR GOTO 명령이 쓰이면 자동적으로 127 보다 큰 값으로 변하는데 이 때 POKE 로 이 값을 다시 127 이하로 해 주면 ONERR GOTO 문의 영향권에서 벗어날 수 있다.

예

```
]LOADEX2
]LIST

4  REM  EX2
5  REM  ***********************
6  REM  * ONERR FLAG          *
7  REM  ***********************
8  REM  .......................S.C.H
9  DIM HIGH(20)
```

```
10  ONERR  GOTO 1000
20 K = 0
30  READ HIGH(K)
40 SUM = SUM + HIGH(K)
50 K = K + 1
60  GOTO 30
90  VTAB 24
100  PRINT "SUM= ";SUM
110  GR : COLOR= 5
120  FOR X = 0 TO K - 1
130  VLIN 39,39 - HIGH(X) AT 2 * X
140  NEXT X
999  END
1000  REM
1010  INVERSE : PRINT " ALL DATA
      READ.....": NORMAL
1015  FOR P = 1 TO 2000: NEXT P
1020  GOTO 100
2000  DATA  1,2,3,4,5,6,7,8,9,10
2010  DATA   23,20,39,20,45,25
```

위의 예제는 데이터를 읽어들여 그 총합을 프린트하고 각 데이터의 크기를 GR 화면상에 막대그래프로 표시하는 문제이다. 데이터의 갯수를 모르므로 30번에서는 결국 OUT OF DATA의 에러가 나게 될 테지만 10번에 써 놓은 ONERR GOTO문의 영향으로 에러 메시지를 보내지 않은 채 1000번으로 가서

```
ALL DATA READ······
```

라는 메시지만 화면에 표시해 준 후, 다시 100번으로 점프해 가게 된다.

100번에서는 총합을 프린트하고 110번부터는 읽어들인 각 데이터를 그래프로 GR 상에 표시하게 되는데 이 때 이 값이 39보다 크면 에러가 나게 된다.

이 경우에는 이에 대한 에러 메시지를 보내고 수행을 정지해야 하는데도 불구하고 끝없이

```
ALL DATA READ·······
```

를 되풀이하여 찍으며 무한히 에러 루우프를 돌게 될 것이다. 이럴 때 1016번에

```
1016 PQKE 216, 0
```

를 넣어 주면 더 이상 ONERR GOTO의 영향을 받지 않게 된다.

그러나 ?ILLEGAL QUANTITY ERROR IN 130이라는 에러 메시지는 다시 나오게 될 것이다.

응용 3 〈ONERR 에러난 문번호〉 $DA. DB

PEEK(219)*256+PEEK(218)

ONERR 시에 에러난 문번호를 알려준다.

ONERR GOTO 명령을 사용할 때 에러난 곳의 문번호를 알고 싶으면 위의 PEEK 문장을 프린트하여 주면 된다. 다음 예는 0으로 나누는 에러가 문번호 30에서 발생하였음을 사용자에게 알게 하도록 하는 프로그램이다.

예

```
]LOADEX3
]LIST

1  REM  EX3
5  REM  *********************
6  REM  * ONERR ERR LINE#   *
7  REM  *********************
8  REM  .................S.C.H
10  ONERR  GOTO 1000
15  HOME
20  FOR I = 100 TO  - 20 STEP  - 10
30  PRINT " INT(100/";I;")= "; INT (100 / I)
40  NEXT
1000  REM
1005  PRINT
1010  PRINT "===============================
      ========="
1020  PRINT " ERROR OCCURRED IN LINE#"; PEEK
      (219) * 256; PEEK (218)
```

응용 4

〈ONERR 에러 코우드〉 $DE

PEEK (222)

ONERR 시 발생한 에러의 종류에 따라 고유의 코우드를 담고 있다.

【콘트롤-C를 눌러 LIST 보려는 것을 막으려면】

프로그램에서 사용자에게 명령을 기다리는 커어서를 내보내 주지 않으면 사용자는 키이보오드를 타이핑할 수 없어서 리스트를 볼 수도 없다. 커어서를 컴퓨터에게 얻어내기 위해서는 리셋 키이나 CTRL-C 키이를 누르는데 우선 **CTRL-C를 무시해 버리는 방법**을 소개한다. 다음 에러 코우드를 보자.

0 = ?NEXT Without FOR
1 = Language Not Available
2 or 3 = Range Error
4 = Write-Protected
5 = End of Data
6 = File Not Found
7 = Volume Mismatch
8 = I/0 Error
9 = Disk Full
10 = File Locked

11 = Syntax Error (DOS)
12 = No Buffers Available
13 = File Type Mismatch
14 = Program Too Large
15 = Not Direct Command
16 = ?Syntax Error (FP)
22 = ?RETURN Without GOSUB
42 = ?Out of DATA
53 = ?Illegal Quantity
69 = ?Overflow
77 = ?Out of Memory
90 = ?Undef'd Statement
107 = ?Bad Subscript
120 = ?Redim'd Array
133 = ?Division by Zero
163 = ?Type Mismatch
176 = ?String Too Long
191 = ?Formula Too Complex
224 = ?Undef'd Function
254 = ?Reenter
255 = Ctrl-C Interrupt

CTRL-C에 해당하는 에러 코우드는 255이다. 그러므로 ONERR 시 PEEK(222)=255이면 여기에 대한 대응책을 마련해 주면 될 것이다.

예

```
1   REM  EX4
5   REM  *********************
6   REM  * ONERR  ERR CODE  *
7   REM  *********************
8   REM ..................S.C.H
10  ONERR  GOTO 1000
20  GR : HOME
100 C =  INT ( RND (1) * 16)
110 X =  INT ( RND (1) * 40)
120 Y =  INT ( RND (1) * 40)
130  COLOR= C: PLOT X,Y
140  GOTO 100
1000  REM
1010  IF  PEEK (222) <  > 255 THEN  END
1020  PRINT " CTRL-C ?": PRINT
1030  PRINT "   YOU ARE FOOL !"
1040  FOR P = 1 TO 2000: NEXT : HOME
1050  GOTO 100
```

이 프로그램은 GR 화면을 16 색깔의 사각점으로 끝없이 PLOT 해 나가는 프로그램인데, 만일 다른 사람이 리스트를 보기 위해 CTRL-C 를 누른다면

```
CTRL-C?
YOU ARE FOOL!
```

이라는 메시지를 보내주고, 이는 아랑곳 않은 채 계속 수행을 하게 될 것이다.

응용 5

〈Read 키이보오드〉

$ C000

49152

PEEK (-16384)

키이가 눌려졌는지의 여부와 가장 최근에 눌려진 키이의 ASCII 값을 알려준다.

아스키(ASCII)[⑥] 코우드 자체는 7비트로 되어 있기 때문에 이들 값은 항상 128보다 작다.

그러나 키이보오드에서 어떤 키이가 눌려지면 −16384번지에는 7비트의 아스키 값과 함께 b_7에 1을 써 넣어 준다("APPLE 어셈블리(교학사)" 참조).

그러므로 **이 때의 −16384번지 값은 128이상이 되어서, 컴퓨터는 키이가 눌려진 것을 알게 되는 것**이다.

그러므로 눌려진 키이의 실제 아스키 코우드 값은 −16383번지의 내용에서 128을 빼주어야 한다.

이 빼주는 작업을 키이보오드 스트로우브[⑦]의 클리어라고 하여서 키이가 눌려진 직후 애플의 모니터 프로그램에서 자동으로 행해 준다.

응용 6

〈Clear 키이보오드〉 $C010

49168

```
POKE-16368, 0
```

키이보오드 스트로우브를 클리어시켜 준다. 비로소 다음 키이 입력이 가능하게 된다.

어떤 키이가 눌려진 다음 그 이후에도 다른 키이의 입력을 받아들일 수 있게 하려면, $C000번지의 비트 7을 0으로 클리어시켜 주어야 하는데 이는 −16368번지를 참조해 줌으로써 행해진다.

우리가 통상 키이를 누를 때는 −16383번지에 그 해당 코우드값이 들어간 직후 모니터 프로그램(ROM 속에 있는)에 의해 자동적으로 −16368번지를 참조해 줌으로써 키이보오드 스트로우브의 클리어가 일어난다.

하지만 BASIC 프로그램상에서 PEEK(−16384)를 써 준 경우에는 반드시 POKE −16368, 0 등으로 스트로우브를 클리어시켜 주어야 한다. 그렇지 않으면 다음 키이 입력을 받아들일 수 없게 될 것이다.

예

```
]LOADEX6
]LIST

5  REM  EX6
```

```
6  REM  ******************
7  REM  * READ,CLEAR KEY  *
8  REM  ******************
9  REM ....................S.C.H
10  HOME
20  VTAB 5: PRINT "    PRESS ANY KEY"
30  PRINT : PRINT "    AND I CAN TELL WHAT
    IT WAS ."
40 X =  PEEK ( - 16384): IF X < 128 THEN 40
50  POKE  - 16368,0
60  VTAB 20
70  PRINT "          IT WAS .....<"; CHR$ (X -
    128);">"
```

40번 문장에서는 어떤 키이가 눌려지기를 기다리면서 끝없이 루우프를 돌고 있다. 그러나 INPUT 문과 달리 화면에 커어서가 나타나지는 않는다.

50번에서는 키이보오드 스트로우브를 클리어시켜 주었고, 70번에서는 눌려진 키이의 아스키 코우드 값(7 bit)만을 뽑아 내기 위해 b_7에 있던 1, 즉 128을 빼준 것이다.

응용 7

〈스피커 신호 보내기〉

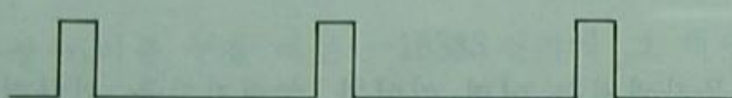
$ C030

49200

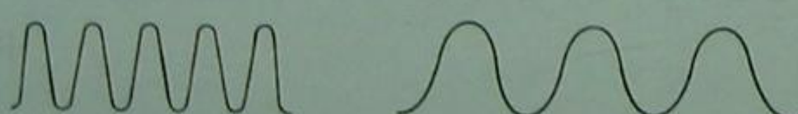
PEEK (-16336)

스피커에 한 번의 펄스 신호를 준다.

−16336 번지를 한 번 참조하게 되면 한 개의 펄스 신호가 스피커를 '틱'하고 울리게 되는데 이것을 연속적으로 보낸다면 일정한 음을 얻을 수 있다.



그러므로 PEEK와 그 다음 PEEK와의 사이가 짧을수록 높은 주파수를 형성하게 되어 높은 소리가 나게 될 것이다.



고주파(높은 소리) 저주파(낮은 소리)

또한 BASIC은 그 속도가 느리므로 아무리 빨리 PEEK(−16336)을 지정해 보았자 높은 음은 낼 수 없다.

예

```
]LOADEX7
]LIST

5  REM  EX7
6  REM  *******************
7  REM  * READ,CLEAR KEY  *
8  REM  *******************
9  REM ......................S.C.H
10  HOME
20  VTAB 10
30  INPUT " ENTER A NUMBER (0-7):";NUM
40  FOR I = 1 TO 100
50 S =  PEEK ( - 16336): FOR D = 1 TO 3 *
   NUM: NEXT
60  NEXT I
70  PRINT : PRINT : GOTO 30
```

30 번에 있는 NUM 은 딜레이 상수로서 40~60 의 FOR~NEXT 루우프가 돌아가는 속도를 또 하나의 FOR~NEXT 문으로 조절하게 된다.

```
FOR D=1 TO 3*NUM : NEXT
```

NUM 이 큰 수일수록 S=PEEK(−16336)이 지정되는 간격이 길어지므로 낮은 소리가 난다. 아예 위의 FOR~NEXT 딜레이 문을 빼버리면 제일 높은 음을 얻을 수 있다.

여기서 S 는 PEEK 한 값을 프로그램의 형식상 담아두기 위해 쓰인 의미없는 변수이다.

응용 8

〈푸쉬버튼 감지〉

$ C061
49249
$ C062
49250

PEEK(-16287)

게임 패들의 버튼 #0를 감지

PEEK(-16286)

게임 패들의 버튼 #1을 감지

-16287, -16286 두 개의 번지는 각각 그 내용이 127보다 크면, 푸쉬버튼 #0, #1이 눌려졌다는 것을 뜻한다(혹은 게임 패들이 연결되지 않았을 경우).

그러므로 이 번지의 값을 PEEK해 보아서 127보다 클 경우는 어떤 동작을 하도록 프로그램해 주면, 버튼을 누름에 따라서 게임의 포탄이 발사되거나 다른 재미있는 동작들을 수행하도록 만들어 볼 수 있는 것이다.

게임 패들이나 조이 스틱이나 그 내부의 회로는 같으므로 이 PEEK문을 함께 이용할 수 있다.

예

```
5  REM  EX8
6  REM  ********************
7  REM  *PUSH BUTTON CHECK*
8  REM  ********************
9  REM ....................S.C.H
10  HOME
```

```
20   VTAB 10
30   GR
40   IF  PEEK ( - 16287) > 127 THEN
     GOSUB 1000
50   IF  PEEK ( - 16286) > 127 THEN
     GOSUB 2000
60   GOTO 40
1000  REM  SUB-->
1020  FOR X = 0 TO 39
1030  COLOR= 1: PLOT X,20
1040  COLOR= 0: PLOT X,20
1045  NEXT X
1050  RETURN
2000  REM  SUB<--
2010  FOR X = 39 TO 0 STEP  - 1
2020  COLOR= 5: PLOT X,22
2030  COLOR= 0: PLOT X,22
2035  NEXT X
2040  RETURN
```

위의 프로그램은 게임 패들의 버튼 #0를 누르면 컬러 1의 사각점이 화면의 왼쪽에서 오른쪽으로, 버튼 #1을 누르면 컬러 5의 사각점이 오른쪽에서 왼쪽으로 달려가도록 하는 프로그램이다.

프로그램을 간결하게 하기 위하여 서브루우틴 1000번과 2000번을 사용하였다.

아무 버튼도 누르지 않으면 GR 화면에는 아무것도 나타나지 않을 것이다.

응용 9

〈그래픽과 TEXT〉

POKE-16304, 0 그래픽 모우드를 선정	$C050 49232
POKE-16303, 0 TEXT 모우드를 선정	$C051 49233

응용 9~응용 12까지는 스크린 소프트 스위치®라고 불리우는 것들인데 화면을 어떤 식으로 나타내느냐 하는 것을 결정할 수 있다. 최초에 다른 지정이 없었다면 화면은 TEXT 모우드로 되어 있다. 즉 −16303번지가 참조되어 있는 것이다. 그러나

```
POKE-16304, 0
```

해 주면 화면은 HGR이나 GR이 사용할 수 있는 그래픽 모우드의 초기단계가 설정된다.

간단한 프로그램에, 위의 POKE명령을 치면 화면에는 TEXT도 그래픽 모우드도 아닌 이상한 선과 점들이 나타날 것이다. 마치 TEXT 모우드 위에 투명한 그래픽 모우드를 얹어 놓은 듯하다.

다음 단계로 해야 할 일은 고해상 그래픽이냐 저해상 그래픽이냐를 결정해 주는 것이다.

응용10

〈저해상, 고해상 그래픽〉

POKE-16298, 0 저해상 그래픽을 선택	$C056 -16298
POKE -16297, 0 고해상 그래픽을 선택	$C057 -16297

일단 그래픽 모우드가 선택되면 2차로 그것이 저해상 그래픽이냐 고해상 그래픽이냐를 결정해 주어야 한다.

위의 POKE문 중 하나를 써서 그것을 선택해 준다. 그러나 이것을 선택했다고 해서 GR이나 HGR 명령을 썼을 때처럼 화면이 클리어(clear)되는 것은 아니다.

GR이나 HGR명령 자체에는 그래픽 선택, 저해상, 고해상 선택 외에도 그래픽 페이지의 선택과 더불어 **그래픽 화면을 클리어 시키는 명령 등을 포함한** 여러 가지 기계어 프로그램이 들어있어, 이것은 마치 하나의 서브루우틴처럼 형성되어 있는 것이다.

응용11

〈페이지 1, 페이지 2〉 $C054

49236

```
POKE -16300, 0
```

$C055

49237

TEXT나 그래픽 화면의 페이지 1을 선택

```
POKE -16299, 0
```

TEXT나 그래픽 화면의 페이지 2를 선택

TEXT 화면이나 저해상 그래픽 화면, 그리고 고해상 그래픽 화면은 각각 두 개의 화면으로 나눌 수 있다. 즉 페이지 1과 페이지 2로 나눌 수 있는데 GR이나 HGR 명령시에는 각각의 페이지 1을 선택하여 보여주고 HGR2 명령시에는 페이지 2를 선택하여 보여주고 있다.

HGR2는 HGR의 4줄의 TEXT만을 대체한 것이 아니라 **실제로 완전히 다른 영역의 어드레스를 사용**하고 있는 것이다(APPLE 메모리 맵 참조).

또한 TEXT의 페이지 1과 Lo-Res의 페이지 1, 그리고 TEXT의 페이지 2와 Lo-Res의 페이지 2는 서로 같은 메모리 영역을 쓰고 있다는 것도 기억해야 할 것이다.

응용12

〈전체 그래픽과 혼합 그래픽〉

$C052
49234

POKE -16302, 0

$C053
49235

화면 전체를 그래픽 모우드로 만든다

POKE -16301, 0

그래픽 화면 밑에 4줄의 TEXT를 만든다

【GR 화면 전체를 쓰는 방법】

GR 명령을 사용하면 통상 밑에 4줄의 TEXT가 나온다. 그러나 만일 당신이 그곳까지를 전부 그림 그리는 데 사용해야 할 필요가 있다면 다음과 같은 명령어로써 해결된다.

```
10 GR
20 POKE -16302, 0
```

그러나 이 명령을 사용한 후에도 밑의 4줄에는 이상한 선들이 그려지는데, 이것은 GR 명령 속에는 밑의 4줄에 대한 클리어 지령은 포함되어 있지 않기 때문이다. 이것을 깨끗이 만들려면 컬러 0을 사용하여 덧칠하는 수 밖에 없다.

예

```
5   REM  EX12
6   REM  ******************
7   REM  * FULL GR        *
8   REM  ******************
9   REM  ....................S.C.H
```

```
20   POKE  - 16304,0: REM <GRAPHICS>
30   POKE  - 16298,0: REM <LO-RES>
40   POKE  - 16300,0: REM <PAGE 1>
50   POKE  - 16302,0: REM <FULL >
70   FOR C = 1 TO 15
80   COLOR= C
90   FOR X = 0 TO 19
100   VLIN 0,47 AT X: VLIN 0,47 AT 39 - X
110   NEXT X
120   NEXT C
130   GOTO 70
```

이 예제는 앞서서 GR을 사용했던 것과는 달리 4개의 POKE 문만을 사용하여 저해상도의 전체 그래픽(full graphics)을 보여준 것이다. 20번에서 그래픽 모우드를, 30번에서 저해상 그래픽을, 40번에서 페이지 1을, 50번에서 전체 그래픽 모우드를 선택하여 주었다. 물론 이들 POKE 문만으로는 화면을 클리어시킬 수 없다.

예 【HGR 페이지 1에서 전체 그래픽을 쓰는 법】

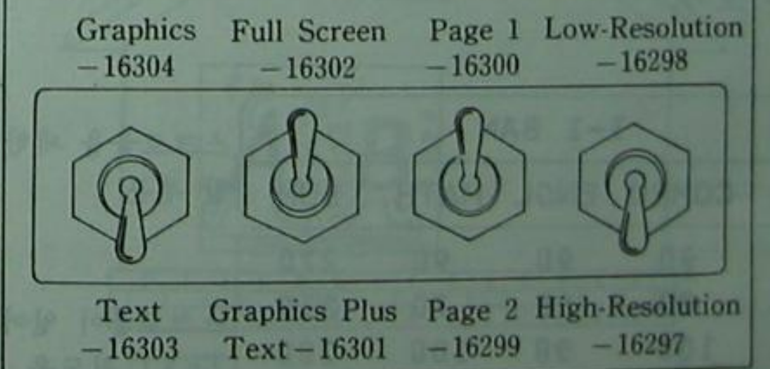
```
]LOADE12'
]LIST

5  REM  EX12'
6  REM  *******************
7  REM  *  FULL  HGR      *
8  REM  *******************
9  REM  ...................S.C.H
```

```
10  HGR
20  POKE  - 16302,0
30  HCOLOR= 3
40  HPLOT 0,0 TO 279,0 TO 279,191 TO
    0,191 TO 0,0
100 OX = 150:OY = 100
110 X =  INT ( RND (1) * 270) + 5
120 Y =  INT ( RND (1) * 180) + 5
125  HCOLOR=  INT ( RND (1) * 6) + 1
130  HPLOT OX,OY TO X,Y
140 OX = X:OY = Y
150  FOR P = 1 TO 300: NEXT
160  GOTO 110
```

위의 프로그램은 HGR2를 쓰지 않고서 HGR 페이지 1 위에 POKE 문을 사용하여 전체 그래픽 모우드(TEXT 와의 혼합이 아닌)로 **추상화를 그려본 것**이다.

이것은 당신이 국민 학교 때 미술 시간에 해본 '구성' 추상화와 비슷한 점이 있을 것이다. 선과 선 사이의 공간을 색으로 메꾸려면 어떻게 해야 할지 연구해 보자.



[스크린 소프트 스위치]

응용13

〈TEXT 윈도우의 좌단 고정〉 $20

```
POKE 32, L
```

TEXT 윈도우의 좌단위치를 L로 고정시킨다.
(L : 0~39)

【TEXT 윈도우란 무엇인가?】

평상시 APPLE의 화면을 사용할 때는 가로 40칸, 세로 24칸을 사용할 수 있었다.

이 때 이 화면 전체는 **스크로울**(scroll) 현상이 일어나는데 이는 **화면의 글자**가 맨 아랫 줄까지 내려오면 맨 윗줄은 **한 칸씩 밀려 올라가게 되는 것**을 말한다. 그러나 어떤 프로그램에 따라서는 화면의 맨 위 몇 줄에는 각 항목의 이름들을 적어놓고, 밑에서 계속 글자들이 찍혀 올라오더라도 이 부분만은 고정되어서 화면에 남아 있어야 될 경우가 종종 생기게 된다.

다음 예를 보자.

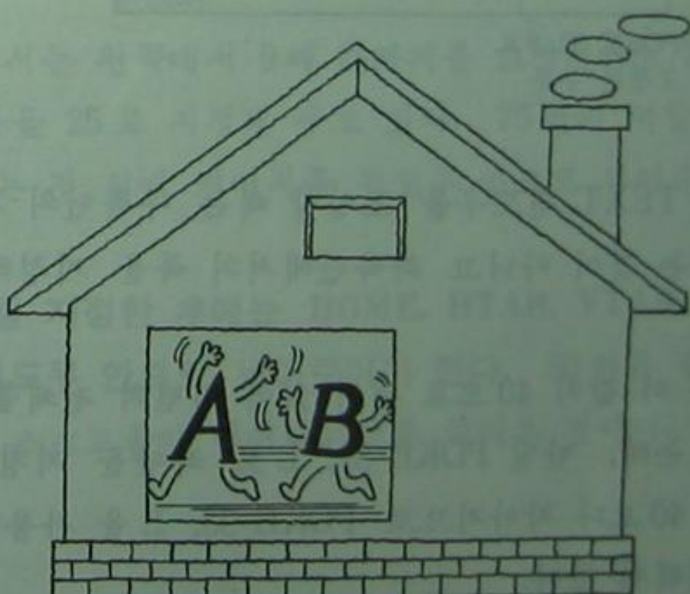
3-1 BAN				
#	COMPU	ENGL	MATH	SUM
1	90	90	90	270
2	80	80	70	230
3	100	90	100	290
:	:	:	:	:

스크로울을 제한해야 할 부분 (3-1 BAN 제목 줄과 # COMPU ENGL MATH SUM 줄)

스크로울이 일어나는 TEXT 윈도우 (1번 이하의 줄)

이것은 어떤 고등학교의 3학년 1반 학생의 컴퓨터, 영어, 수학의 성적표인데, 22명의 학생(제목에서 두 줄을 차지하므로)을 넘어선다면 23번째 학생이 맨 밑줄에 찍히지만 맨 윗줄이 한 줄 없어질 것이다. 또 그다음 학생이 찍히게 되면 COMPU. ENGL. MATH. SUM. 등의 제목마저 없어지게 된다.

이렇듯 어떤 데이터가 아닌, 변할 필요가 없는 부분은 스크롤이 안 일어나도록 고정시키는 것이 요구되며, 이것을 4개의 POKE문으로 각각 좌단, 좌단에서 우측단까지의 폭, 상단, 그리고 하단을 고정시켜서 그 고정된 부분을 제거한 나머지를 TEXT 윈도우라고 부른다.

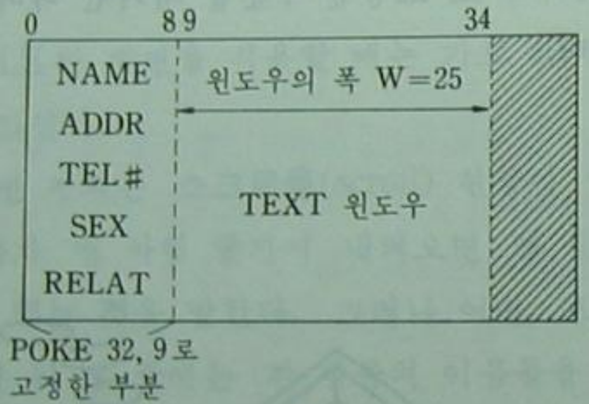


응용14

〈TEXT 윈도우의 폭 고정〉 $21

```
POKE 33, W
```

좌단에서의 윈도우 폭을 W로 고정시킨다.
(W : 40~1) W+1부터 윈도우 시작



APPLE의 TEXT 윈도우를 조절할 때는 우측단의 자리를 직접 지정해 주는 것이 아니고 좌측단에서의 폭을 지정해 주게 된다.

평상시에는 이 값이 40으로 되어 있어 화면의 전체를 쓸 수 있도록 되어 있는데, 만일 POKE 32, L로 좌단을 지정해 준다면 폭은 자연히 40보다 작아지므로 POKE 33, L을 사용하여 새로이 폭도 지정해야 한다.

만일 32번지와 33번지를 동시에 지정해 주지 않으면 에러가 날 것이다.

예 앞의 그림과 같이 화면 윈도우의 폭이 25가 되도록 윈도우를 지정해 보자.

```
]LIST

10  HOME
20  VTAB 5: PRINT "NAME:"
30  VTAB 8: PRINT "ADDRESS:"
40  VTAB 11: PRINT "SEX:"
50  VTAB 14: PRINT "REFERENCE"
60  POKE 32,9
70  POKE 33,25
75  HOME
80  FOR I = 1 TO 300: PRINT "WINDOW. ";:
    NEXT I
```

60번에서는 왼쪽에서 9째 칸까지를 고정시키고 70번에서는 윈도우의 폭을 25로 지정해 주고 있다. 75번의 역할은 화면을 클리어시키는 것 외에 커서를 윈도우 안으로 넣어주는 역할을 한다.

윈도우를 지정한 후에는 HOME, HTAB, VTAB 등을 써서 커어서를 윈도우 안으로 넣어주어야 한다. 80번은 실제로 윈도우 안에서만 스크로울이 일어나는지를 알아본 것이다.

응용15

〈TEXT 윈도우의 상단 고정〉 $22

```
POKE 34, T
```

윈도우 윗쪽 고정부분을 T번째 줄로 지정한다.
(T : 0~23)

TEXT 윈도우를 지정할 때 가장 많이 사용되는 것이 바로 상단번지인 34번지인데, 이는 스크로울이 아래에서 위로 일어나기 때문이다. 그러므로 화면에 찍혀야 할 데이터는 계속 찍혀 올라가지만 맨 위의 제목들은 POKE 34, T를 사용하여 고정시키는 것이다.

만일 상단 TEXT 윈도우를 해제시키려면 원래 가졌던 값 0을 POKE해 준다(POKE 34, 0).

지금 우리가 다루고 있는 번지는 제로 페이지(zero page)의 번지들인데 이곳은 RAM 영역이므로 그 내용을 사용자가 바꿀 수 있다.

이렇듯 사용자가 컴퓨터의 내부 시스템까지 만져볼 수 있다는 것이 퍼스널 컴퓨터의 큰 묘미인 것이다.

예 키이를 누를 때마다 학생 한 명의 성적이 찍혀나가도록 프로그램하라. 단 맨 위 세 줄은 제목과 과목을 써야 하므로 스크로울이 일어나지 말아야 한다(학생수 30, 과목 COMP. ENGL. MATH).

```
]LIST

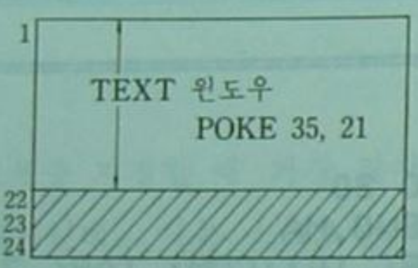
5  REM  EX15
6  REM  ********************
7  REM  *TEXT WINDOW(UPPER)*
8  REM  ********************
9  REM .....................S.C.H
10  HOME
20  HTAB 11: PRINT "<EXAMINATION REPORT>"
30  PRINT "#        COMP.        ENGL.        MAT
    H."
33  PRINT "======================================
    ======"
35  POKE 34,3
40  FOR I = 1 TO 30
50  READ N,COM,ENG,MA
60  PRINT N; TAB( 10);COM; TAB( 22);ENG; TAB(
    33);MA
70  GET R$
80  NEXT
90  DATA  1,88,99,56,2,44,88,96,3,67,85,78
100  DATA  4,78,67,98,5,87,56,38,6,88,78,90
110  DATA  7,45,67,89,8,78,78,98,9,87,97,56
120  DATA  10,87,67,98,11,67,89,87,12,45,67,
    87
130  DATA  13,56,87,98,14,56,65,66,15,78,77,
    88
140  DATA  16,78,67,77,17,78,88,95,18,77,87,
    63
150  DATA  19,66,88,87,20,67,87,94,21,56,57,
    87
160  DATA  22,45,65,87,23,55,63,82,24,63,73,
    95
170  DATA  25,86,56,78,26,77,88,67,27,56,73,
    84
180  DATA  28,93,85,86,29,99,74,87,30,67,89,
    99
```

응용 16 〈TEXT 윈도우의 하단 고정〉 $23

```
POKE 35, B
```

TEXT 윈도우의 맨 아랫단을 B로 지정한다.

(B : 1~24)



35번지의 값은 원래 24로 지정되어 있지만 윗 그림처럼 아래 세 줄은 스크로울이 일어나지 않게 하려면 POKE 35, 21 하면 된다. 4개의 TEXT 윈도우 파라미터[5]들을 알아보려면 다음과 같이 해보자.

```
10 FOR WIN=32 TO 35
20 PRINT WIN, PEEK(WIN)
30 NEXT
```

원래의 32, 33, 34, 35번지 값은 각각 0, 40, 0, 24로 되어 있을 것이다.

응용17

〈첫 줄만 LIST 시키기〉

```
POKE 2049, 0
```

LIST 시켰을 경우 프로그램의 첫번째 줄만 보여 주게 한다.

【상대방이 첫 줄만 LIST 할 수 있게 만드는 법】

앞에서도 몇 가지 방법이 있었으나 이번에는 LIST를 치면 오직 BASIC 프로그램의 첫 줄만 리스트되게 하는 방법이다. 프로그램의 첫 줄에 다음과 같이 REM 문을 쓰고 POKE를 해보자. 리스트시키려 하면 "뭘봐!" 하는 메시지만 찍힐 것이다.

당신의 프로그램 리스트를 보려는 사람에게 약올리는 방법의 하나이다.

```
1   REM   " MUEL BOA ! "
5   REM  EX17
6   REM  *********************
7   REM  *  MUEL BOA !       *
8   REM  *********************
9   REM .....................S.C.H
20   POKE 2049,0
30   HOME
50   VTAB 15: HTAB 15
60   PRINT "CAN YOU LIST ?"
70   END
```

응용 18

〈N개 화일씩 CATALOG 시키기〉

```
POKE 44452, N+1
POKE 44605, N
```

CATALOG 명령시에 N개의 화일 이름을 화면에 보여준 후 멈추고, 키이를 누르면 다시 N개의 화일 이름을 찍어낸다.

이 명령은 두 개의 POKE문을 써야 하는데 여기서 N은 한 번 키이를 누를 때마다 화면에 프린트 되기를 원하는 화일의 갯수이다. 예를 들면 6개씩 CATALOG를 보고 싶으면

```
POKE 44452, 7:POKE 44605, 6
```

하면 된다.



응용19

〈한 화일씩 CATALOG 시키기〉

```
POKE 44599, 234
POKE 44600, 234
```

CATALOG를 치면 다시 키이를 누를 때마다 한 개씩만 화일 이름을 보여준다.

CATALOG를 치면 보통 여러 개의 화일이 빨리 찍혀서 올라가 버린다. 만일 하나씩 천천히 살펴 보고 싶은 경우가 있다면 위의 2개의 POKE명령을 쓰면 된다.

아무 키이를 한 번씩 누를 때마다 한 줄씩 화일 이름이 리스트될 것이다.



응용 20

〈전 화일 이름을 한번에 CATALOG 시키기〉

```
POKE 44596, 234
POKE 44597, 234
POKE 44598, 234
```

중간에 멈춤없이 계속 CATALOG 시킨다.

이 POKE 명령을 일단 쳐 놓으면, 다음에 CATALOG를 칠 때마다 맨 처음 화일부터 맨 나중 화일 이름까지가 계속 스크로울되어 올라가며 화면에 찍히게 된다.

실질적으로 앞쪽 화일 이름들은 미처 볼 수가 없으므로, 나중에 들어간 화일 이름들을 빨리 알아보고 싶을 때 사용된다.

당신이 만일 어떤 프로그램을 작성하고 있다면 그 중간 중간의 작성된 프로그램에는 번호를 붙인 화일 이름, 즉 HAHN 1, HAHN 2, …, HAHN 7 등으로 이름을 붙여서 SAVE 시킬 수 있는데, 사실상 당신이 다음 번에 참고할 화일 이름은 마지막 이름인 HAHN 7이라고 볼 수 있으므로 이러한 식으로 프로그램을 작성할 때 위의 POKE를 한번만 해두면 편리할 것이다.

응용 21

〈고해상 그래픽 화면을 바탕색으로 칠하기〉

$F3F2

```
CALL-3086
```

HGR, HGR 2 화면을 바탕색인 HCOLOR=0 으로 칠해 버린다.

이 명령어는 고해상도 그래픽 영역에 전부 0을 넣어주는 효과를 내는 것인데, 이미 HGR 이나 HGR2 가 선언되어 있지 않더라도 그 내부에서는 동작을 한다.

따라서 HGR 화면을 보여주고 나서 CALL-3086 하면 화면이 깨끗이 지워지지만, TEXT 모우드나 GR 모우드에서 CALL-3086 을 한다고 해도 화면에는 보이지 않을 것이다.

이 서브루우틴은 또한 어셈블리 모우드에서

```
JSR $F3F2
```

하여, 고해상도 화면을 클리어시키는 데에 많이 사용된다.



응용22

〈고해상 그래픽 바탕 칠하기〉 $F3F6

```
CALL-3082
```

가장 최근에 HPLOT된 색깔로 고해상 그래픽 화면 전체를 순간적으로 칠해 버린다.

보통 BASIC으로 HGR 화면을 칠하려면 상당히 많은 시간이 걸린다. 예를 들어 컬러 2로 칠하려면

```
100 HGR:HCOLOR=2
110 FOR X=0 TO 279
120 HPLOT X,0 TO X,159 :NEXT
```

하는데 시간이 7초 정도는 걸리게 된다. 그러나 다음과 같이 하면 순식간에 칠할 수 있다.

예

```
100 HGR:HCOLOR=2
110 HPLOT 0,0
120 CALL-3082
```

1초도 걸리지 않게 되는 것이다.

응용 23

〈소프트웨어 리셋〉 $FA62

```
CALL-1438
```

프로그램상에서 리셋을 누른 효과를 낸다.

이 번지는 실제로 리셋 키이가 눌려졌을 때에도 수행하게 되는 서브루우틴 시작 번지로서 프로그램 중간에서 이것을 CALL 하면 리셋이 걸리게 된다.

따라서 프로그래머가 원하는 위치에서 리셋이 자동으로 걸리도록 프로그램할 수 있는 장점이 있다.

다음과 같은 프로그램으로 확인을 해보자.

```
100 FORI=1 TO 10
110 PRINT I
120 IF I=5 THEN CALL-1438
130 NEXT I
```

위의 루우프는 I가 5가 되면 리셋이 걸리게 되므로 1, 2, 3, 4, 5까지만 프린트할 것이다.

응용24

〈DOS를 부팅시키기〉

$FA 87

```
CALL-1370
```

PR#6과 같이, DOS를 부팅시킨다.

직접 방식이나 또는 프로그램상에서 CALL-1370을 만나게 되면 DOS의 부팅이 일어난다.

```
100 HOME
110 INPUT "BOOT? (Y/N)";B$
120 IF B$="Y" THEN CALL-1370
130 END
```

[참고] 부팅(booting)

디스크 드라이브를 사용하기 위해서는 '디스크 오퍼레이팅 시스템'이라고 불리우는 특별한 프로그램이 컴퓨터 메모리 내에 있어야 한다. 그냥 짧게, 'DOS'라고 부르는 디스크 오퍼레이팅 시스템은 디스크에 관련된 모든 움직임을 통제하게 되는데, 이 DOS를 컴퓨터의 메모리에 집어 넣어주는 작업을 가리켜 'DOS를 부팅'한다고 말한다.

응용 25

〈레지스터 값의 표시〉

$FAD7

CALL-1321

BASIC 상에서 5개의 레지스터 값을 알아볼 수 있다.

어셈블리 어(語)⑩로 된 프로그램을 BASIC의 서브루우틴으로 쓰는 경우 등에서는 어떤 때 그 레지스터 값들이 얼마인가를 알아볼 경우가 생긴다.

이 때 CALL -1321 하면 화면에는

A = 1A X = 00 Y = 00 P = 34 S = B7

등으로 어큐뮬레이터, X 레지스터, Y 레지스터 프로세서 상태 레지스터, 스택 포인터의 현재 값이 찍히게 된다.

그러나 그 후에도 프럼프트 문자는 계속]로 남아 있으며 BASIC 모우드가 그대로 유지된다.



응용 26

〈APPLE][를 표시하기〉 $FB60

CALL-1184

화면을 클리어시키고 APPLE][라 찍는다.

이 부분은 APPEL의 스위치를 켤 때 자동으로 수행되는 서브루우틴의 하나이다. 이 서브루우틴은 화면을 클리어시키고 TEXT 화면이 윗쪽에 A, P, P, L, E,], [의 글자에 해당하는 화면 표시의 아스키 코우드 값을 넣어주는 작업을 한다.

이 루우틴의 문자 데이터 값만 바꾼다면 APPLE][대신에 당신의 이름이 찍히게 할 수도 있다. 그러나 이 서브루우틴은 F800 번지부터 시작되는 모니터 ROM 안에 들어 있으므로 프로그램을 하여 고칠 수는 없다. 즉 ROM Writer⑪(롬 라이터 : ROM의 내부에 정보를 써 넣는 기구)가 필요하게 된다. 이것을 하는 방법에 대해서 알아보자.

【APPLE][대신에 당신의 이름이 찍히게 하려면】

이 작업을 하려면 ROM Writer가 필요하다. 또한 EPROM⑫(2716, 2732 등)을 사용하고 있는 APPLE이어야 한다. 우선 APPLE 내부에 꽂혀 있는 모니터 ROM의 내용을 RAM 영역(예컨대 $

4800 번지부터)으로 옮긴다.

이것은

```
*4800<F800. FFFF M
```

으로써 이루어진다. 여기서는 2K 용량을 가진 2716 ROM에 $ 4800~$4FFF 내용을 ROM Writer로 복사해 주는 방법을 사용할 것이다.

위의 모니터 명령어 M에 의해 $4800번지부터 $4FFF번지까지의 RAM 영역에 Apple 모니터 ROM의 내용이 그대로 복사된다. 모니터 ROM의 $FB09~$FB10의 8개 장소에는 'APPLE][' 에 해당하는 문자의 아스키(ASCII) 코우드가 들어 있다. 그러므로 현재 RAM 영역 $4B09~$4B10에도 똑같은 아스키 코우드가 들어있게 될 것이다. 그 내용은 '$C1, $D0, $D0, $CC, $C5, $A0, $ $DD, $DB'이다. 자, 이것은 RAM 영역에서는 얼마든지 바꿀 수 있다. 물론 이 경우에 8자의 문자데이터까지만 변경할 수 있다.

그러면 이제 화면 표시용 아스키 코우드표(부록 참조)를 보고 당신 이름에 해당하는 데이터들을 골라보자. (8글자)

예를 들어 Apple의 스위치를 켤 경우 화면 위에 "SOOCHAN!" 이라고 나오게 하고 싶으면 해당 아스키 코우드인 $D3, $CF, $CF, $C3, $C8, $C1, $CE, $B1을 $4B09번지부터 차례로 넣어 준다. 그 방법은 모니터 모우드에서,

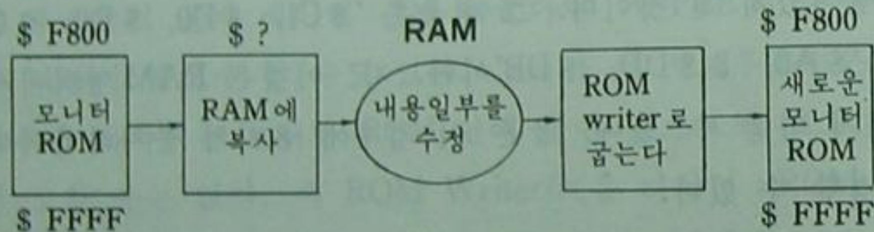
```
*4B09:D3 CF CF C3 C8 C1 CE B1
```

을 타이핑한 후 리턴 키이를 눌러 주면 된다.

제대로 들어갔는지 확인해 본 후 $4800~$4FFF의 내용을 다른 ROM(2716)에다 기록시키면 된다. 이 때 ROM Writer가 필요하게 되는데 그리 비싸지 않은 가격에 시중에서 구할 수 있다.

각각의 롬 라이터의 사용법은 조금씩 다르나, 대개 2716 2732까지는 사용할 수 있도록 되어 있다. ROM에 그 내용을 다 기록(write)시켰으면(보통, ROM을 굽는다고 말함) Apple의 전원을 끄고 새로이 만들어진 ROM을 원래 있던 모니터 ROM과 교체시켜 삽입한다.

이제 Apple의 전원을 넣어 보자. 당신의 이름이 당당하게 화면에 나타날 것이다.



응용 27

〈& 벡터 사용법〉 $3F6~$3F7

```
POKE 1014, L
POKE 1015, H
```

이 두 개의 번지에는 프로그램이나 직접 방식에서 &(Ampersand: 앰퍼샌드)를 만났을 때 **점프해 갈 어드레스가 들어있다.** 어떤 기계어 서브루우틴을 자주 사용해야 할 경우에 프로그램의 앞에서 일단 POKE 문으로 그 어드레스만 지정해 주면, **다음 번에 &를 만날 때마다 그 어드레스로 점프해 가게** 되는 편리한 기능이다.

이것을 사용한 좋은 예로, 시중에 돌아다니는 한글 프로그램을 사용할 때 &를 쳐 주어야 동작을 하게 되어있는데 PEEK(1014), PEEK(1015) 해 보면 각각 255와 12로 되어 있다.

이 두 어드레스를 붙여서 계산해 보면

$$255 + 12 \times 256 = 3327$$

이 된다. 따라서 이 때 &는 CALL 3327의 효과를 내는 것이다.

【&로 BASIC 명령어를 대체하는 법】

예 1 LIST → &

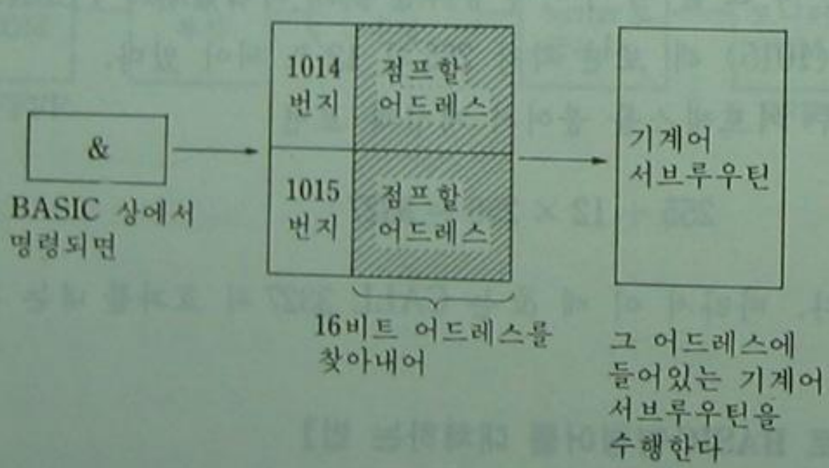
```
10 POKE 1014, 165
20 POKE 1015, 214
```

이제 이 프로그램을 RUN시킨 후 &를 치면 프로그램이 LIST될 것이다.

예 2

```
RUN → &
10 POKE 1014, 18
20 POKE 1015, 217
30 FOR I=1 TO 10
40 PRINT "***"
50 NEXT
```

이 프로그램을 RUN시킨 후 &를 치면 프로그램이 다시 RUN될 것이다.



[&가 수행되는 과정]

응용28

〈CTRL-Y 벡터 사용법〉

$ 3F9
$ 3FA

```
POKE 1017, L
POKE 1018, H
```

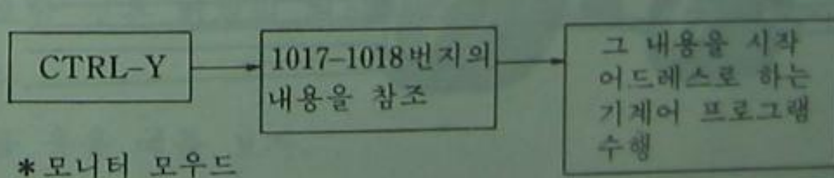
모니터(*) 모우드에서 CTRL-Y를 누르면 위의 두 번지가 담고 있는 어드레스로 점프해 간다.

이 기능은 모니터 모우드에서만 가능하기 때문에 실제로 POKE를 사용하기 보다는 $ 3F9·3FA 번지에 16진 어드레스를 직접 넣어주고 CTRL-Y를 이용하는 것이 보편적이다.

예를 들어 APPLE][를 찍는 $ FB60 서브루우틴을 CTRL-Y에 의해서 동작하게 하려면

```
*3F9:60 FB
```

를 한다음, 모니터 모우드에서 CTRL-Y를 누르면 화면이 클리어되고 APPLE][라고 찍히게 될 것이다.



응용 29

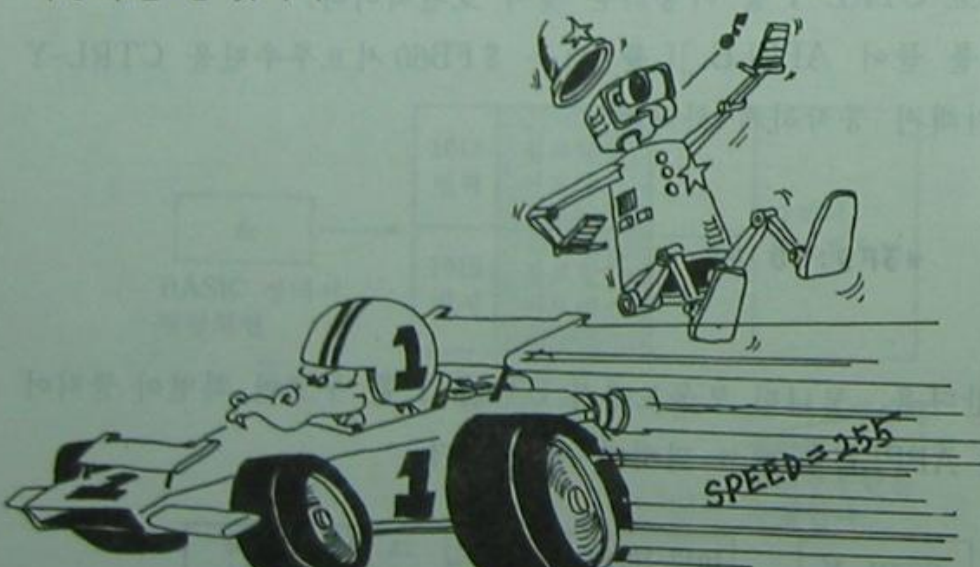
〈SPEED의 데이터〉 $F1

```
PEEK (241)
```

(256 — 현재 SPEED값)이 담겨 있다.

현재의 SPEED값이 255라면 241번지에는 256-255한 1이 담겨 있고, SPEED를 100으로 지칭한 뒤 PRINT PEEK(241)하면 256-100한 156이 찍힐 것이다.

프로그램 중간에서 SPEED값에 대한 데이터를 얻고자 할 때 사용하면 될 것이다.



응용 30

〈TEXT 라인 지우기〉

$ FC9C

CALL-868

커어서의 위치에서 그 줄의 오른쪽 끝까지를 지운다.

이것을 잘만 사용하면 아주 유용한 서브루우틴인데 실제로 국내에서는 거의 사용되고 있지 않은 듯하다.

우선 이것이 어떻게 동작하는지를 확실히 알기 위해 다음 프로그램을 RUN 시켜 보자.

```
100 HOME
110 FOR I=1 TO 20
120 PRINT I;"**********":NEXT I
130 VTAB 3: HTAB 5: CALL-868
```

이 프로그램은 20줄의 *무늬 띠를 찍고, 다시 위에서 3째줄, 오른쪽으로 5번째의 자리부터 그 오른쪽 끝까지를 CALL-868에 의해서 지워버렸다.

중요한 것은 현재의 커어서 위치를 항상 염두에 두어야 한다는 점이다.

다음 응용 예를 보자.

예

```
]LOADEX30
]LIST

1  REM  EX30
2  REM  *********************
3  REM  * CLEAR ONE LINE    *
5  REM  *********************
9  REM ....................S.C.H
100  HOME
110  VTAB 5: PRINT "ENTER YOUR NAME ONE
     BY ONE."
120  VTAB 10
130  FOR I = 1 TO 3
140  PRINT I".";: INPUT "NAME:";NA$(I)
150  PRINT "..............O.K(Y/N)?";:
     GET Y$: HTAB
     1: CALL  - 868
160  IF Y$ = "N" THEN 140
170  NEXT I
180  PRINT : PRINT
185 FULL$ = NA$(1) + " " + NA$(2) + " "
    + NA$(3)
190  PRINT "**  YOUR FULL-NAME IS :";
     FULL$
```

이 프로그램은 당신의 이름 석자를 차례로 넣어 주면 컴퓨터가 다시 완전한 이름을 프린트해 주는 프로그램인데, 150번 문장은 이름을 입력시키다가 잘못 쳐 넣었을 경우에 대비해서 그 자리에서 즉시 수정할 수 있도록 해 주는 문장이다.

그러나 이 OK(Y/N)?라는 메시지는 화면에 계속 남아있을 필요가 없으므로 HTAB1과 CALL-868을 사용하여 사용자가 Y나 N을 친 직후에 사라지도록 만들었다.

응용31 〈커어서의 수평 위치〉

$24

PEEK (36)

커어서의 현재 수평 위치(0~39)를 담고 있다.

36번지의 내용은 계속 0~39 사이를 왔다 갔다 하며 변하고 있다. 이 번지를 PEEK해 봄으로써 현재의 커어서 수평 위치를 알아내고, 또 POKE해 줌으로써 원하는 곳에 커어서를 옮길 수도 있다.

HTAB은 1~24의 범위이지만 36번지 내용은 HTAB값에서 1을 뺀 값이 될 것이다. 그냥 고정적으로 가로 위치를 지정한다면 HTAB을 사용하는 것이 편리하지만 이 위치를 계산에 사용하는 변수로 사용할 때에는 PEEK(36)과 POKE 36, X를 쓰는 것이 편할 때도 있다.

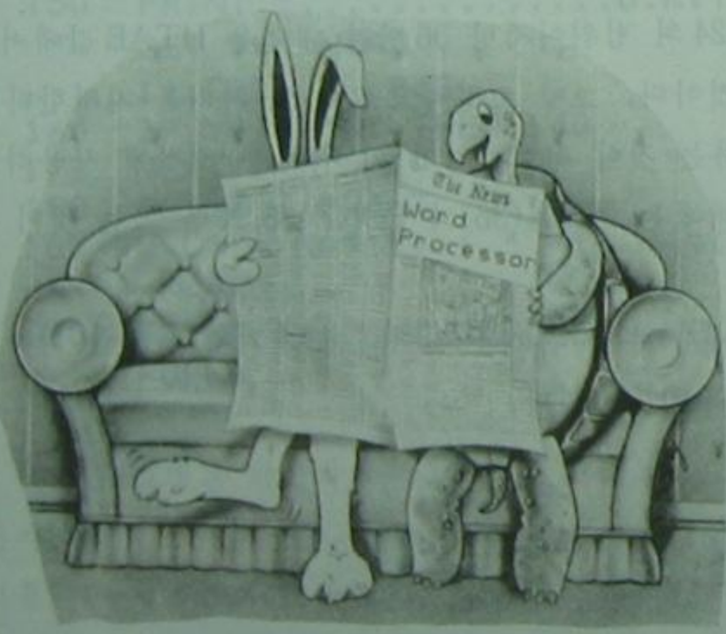
즉 현재의 커어서 수평 위치를 알아내어 거기에 상수를 더하거나 빼는 등에 사용할 수 있다.

예

```
]LOADEX31
]LIST

1  REM  EX31
5  REM  ********************
6  REM  * HORIZ-CURSOR     *
7  REM  ********************
9  REM ..................S.C.H
```

```
100  HOME
110  VTAB 20
120  FOR I = 1 TO 20
125  HTAB I
130  PRINT "*"; PEEK (36)
140  NEXT
```



응용32

〈커어서의 수직 위치〉 $25

PEEK (37)

커어서의 현재 수직 위치 (0~23)를 담고 있다.

37 번지의 내용도 커어서의 위치가 변화함에 따라 계속 변하고 있다. VTAB은 1~24 인데 37 번지는 0~23 인 점에 유의해야 한다.

이 번지를 PEEK 해 봄으로써 그 당시의 커어서 수직 위치를 알아낼 수 있고 POKE 37, X 함으로써 커어서의 수직 위치를 결정할 수도 있다.

다음 예에서는 37 번지의 내용을 PEEK 하여 감소시킨 후 다시 POKE 해 주는 테크닉을 보여주고 있다.

```
10  REM  EXER32
20  REM  LIFT THE CURSOR
30  REM ...................S.C.H
100  HOME
104  VTAB 5
105  INVERSE
110  VTAB 24
115  NORMAL
120  FOR A = 1 TO 22
130  POKE 37, PEEK (37) - 2
140  HTAB 10: PRINT ".........^........."
150  NEXT
```

예

```
]LOADEX32
]LIST

5  REM  EX32
6  REM  *********************
7  REM  *  VERT-CURSOR      *
8  REM  *********************
9  REM ....................S.C.H
110  HOME
120  FOR I = 1 TO 22
130  PRINT "VERTICAL CURSOR POSITION:";
     PEEK (37)
140  NEXT
```

HOME 후에 FOR~NEXT 루우프의 인덱스 I가 1~22로 변하므로 수직 위치도 첫째 줄에서 22번째 줄로 변화해 가지만 37번지의 값은 0~21로 변해가는 것을 알아볼 수 있다.

커어서의 수평, 수직 위치를 알아야 할 경우는 많지만, 그 중에서도 중요하게 응용되는 곳이 바로 TEXT WINDOW(텍스트 윈도우)를 설정하고 사용할 때이다.

또, 일단 윈도우 밖으로 나간 커어서는, 다시 윈도우 안으로 들어오게 하기 위해서 POKE, HOME, HTAB, VTAB 문장 등을 사용해야 할 것임을 잊지 말자.

응용 33

〈INVERSE, FLASH, NORMAL〉 $32

```
POKE50, 63 (INVERSE)
POKE 50, 255 (NORMAL)
POKE 50, 127 (FLASH)
```

APPLE 이 화면에 문자를 표시할 때 50 번지의 내용을 참조해서 그것이 63 이면 INVERSE, 127 이면 FLASH, 255 이면 NORMAL 로 찍게 된다. 이 세 가지 숫자를 가만히 살펴 보면

$$
\begin{array}{rcl}
 & & b_7 \qquad\qquad b_0 \\
63 & \rightarrow & 0\,0\,1\,1\,1\,1\,1\,1 \\
127 & \rightarrow & 0\,1\,1\,1\,1\,1\,1\,1 \\
255 & \rightarrow & 1\,1\,1\,1\,1\,1\,1\,1
\end{array}
$$

로써, b_7 과 b_6 의 값이 00 이냐 01 이냐 11 이냐에 따라서 그 기능이 달라지게 된다.

APPLE 은 문자의 ASCII 코우드에 이 50 번지의 내용을 AND* 시켜서, 그 결과를 화면 표시용의 ASCII 코우드로 하고 있는 것이다.

어셈블리 어로 프로그램할 때는 유용한 번지이다.

[예]

```
5  REM  EX33
6  REM  ********************
7  REM  * INVER,FLASH,NORM *
8  REM  ********************
9  REM .....................S.C.H
110  HOME
120  POKE 50,63
130  FOR I = 1 TO 8
135  PRINT "THIS IS INVERSE."
136  NEXT
137  FOR P = 1 TO 900: NEXT
140  POKE 50,127
150  FOR I = 1 TO 8
160  PRINT "THIS IS FLASH."
170  NEXT
180  FOR P = 1 TO 900: NEXT
190  POKE 50,255
200  FOR I = 1 TO 8
210  PRINT "NORMAL AGAIN."
220  NEXT
```

[참고] 아스키 코우드와 50번지 내용의 AND

예를 들어 'A'라는 문자가 들어왔을 때 그 코우드는 $C1이다. 이것은 2진수 '11000001'인데 여기에 50번지의 내용을 AND시키면

'A'	11000001	11000001	11000001
50번지 내용	00111111 (63)	01111111 (127)	11111111 (255)
AND한 결과의 A	00000001 (Inverse $01)	01000001 (Flash $41)	11000001 (Normal $C1)

(자세한 내용은 Apple 어셈블리(교학사) p. 155 참조)

응용 34 〈APPLE 기종을 판별하는 법〉

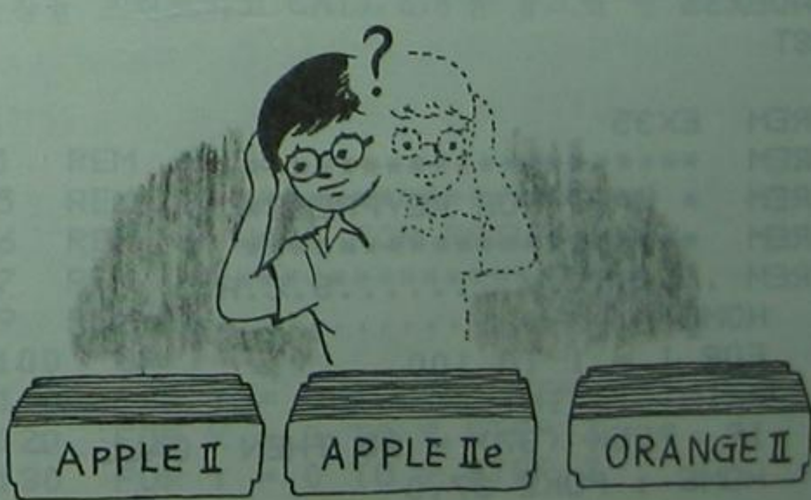
```
PEEK (-637)
```

이 값에 따라 APPLE의 기종을 판별할 수 있다.

PEEK(−637)=255 이면
APPLE Ⅱe 이고

PEEK(−637)=223 이면
APPLE Ⅱ 이거나 APPLE Ⅱ+ 이고

PEEK(−637)=234 이면
Franklin Ace 등의 APPLE 호환성 기종이다.



응용35

〈키이 누를 때까지 기다림〉 $FD0C

CALL-756

사용자가 아무 키이나 누르기 전까지는 프로그램이 진행되지 않는다.

화면에 글씨를 써 주고 사용자가 이것을 읽을 시간을 주기 위하여 잠시 프로그램의 진행을 멈출 필요가 있다. 이 때 그 부분에 가서 CALL-756을 끼워 두면 커어서가 깜박이며 키이가 눌려질 때까지 기다리게 되는 것이다.

다음 프로그램은 1에서 100까지 수의 제곱을 프린트하는 것이다. 중간 중간 정지하기 위해 CALL-756을 넣었다.

예

```
]LOADEX35
]LIST

1  REM  EX35
5  REM  ********************
6  REM  * WAIT FOR KEYPRESS*
7  REM  ********************
9  REM  ...................S.C.H
100  HOME
120  FOR I = 1 TO 100
130  PRINT I, INT (I ^ 2)
140  IF  PEEK (37) = 23 THEN  CALL  - 756:
     HOME : POKE 37,0
150  NEXT
```

응용 36

〈리턴 키이를 누를 때까지 기다림〉 $FD5A

```
CALL-678
```

사용자가 리턴 키이를 누를 때까지는 프로그램이 진행되지 않는다.

이 기능도 CALL-756과 비슷하게 사용되는데, 누르는 키이가 리턴 키이로 한정되어 있다는 것이 다르다. 그러므로 프로그램이 멈춰질 필요가 있는 곳에

```
"PRESS <RETURN> TO CONTINUE..."
```

등의 문장을 프린트하고 CALL-678을 넣으면 될 것이다.

예

```
1   REM  EX36
5   REM  *********************
6   REM  * WAIT FOR RETURN   *
7   REM  *********************
9   REM  ..................S.C.H
100  GR
110  FOR I = 0 TO 30 STEP 3
120  FOR X = 0 TO 9 STEP 2
130  FOR Y = 0 TO 9 STEP 2
140  COLOR= X + 1
```

```
150  PLOT I + X,I + Y
160  NEXT Y: NEXT X
170  HOME : PRINT "PRESS <RETURN> TO
     CONTINUE..........."
180  CALL  - 678
185  HOME
190  NEXT I
```

170~180번 문장은 CALL을 사용하지 않고 INPUT문으로 다음과 같이 바꿀 수도 있다.

```
170 HOME
180 INPUT "PRESS <RETURN> TO
    CONTINUE........";R$
```

INPUT문을 만났을 때 그냥 리턴 키이를 눌러주면 INPUT문의 변수 R$은 아무 값도 갖지 않게 된다. (null character)

또는 GET와 PRINT를 써서 다음과 같이 할 수도 있다.

```
180 PRINT "PRESS ANY KEY TO CONTINUE
    ......;:GET R$
```

이 경우는 어떤 키이를 눌러도 상관없다.

응용 37

〈리셋 키이를 거부하는 법〉 $3F4

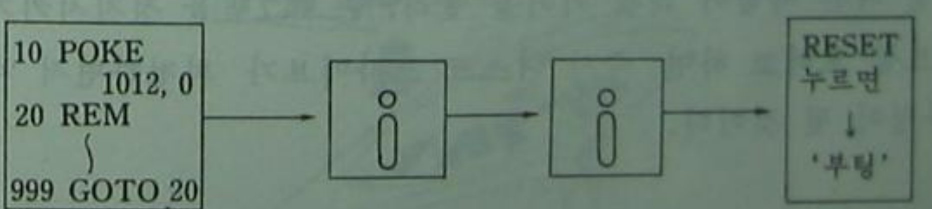
```
POKE 1012, 0
```

리셋 키이를 누르면 DOS를 부팅시킨다.

【RESET 키이를 눌러 프로그램 리스트 보려는 것을 막는 법】

당신이 애써 짜 놓은 프로그램을 다른 사람이 훔쳐보고 그대로 모방한다는 것은 기분 나쁜 일이다. 이 때 이 프로그램이 디스크를 사용하는 경우라면 프로그램의 맨 앞에 POKE 1012, 0를 써 준 후, 프로그램을 SAVE할 때는 그냥 SAVE하지 말고 이 프로그램을 그리팅 프로그램(greeting program)으로 이니셜라이즈[14] (INIT)시킨다면, 웬만한 기술을 가진 사람이 아니고는 프로그램 리스트를 볼 수 없을 것이다.

물론 이 프로그램은 END나 STOP문으로 끝나서는 안 된다. (커어서를 상대편에게 보여 주므로) 따라서 무한루우프를 돌게 하거나, 다시 프로그램 처음으로 돌아가게 해야 할 것이다.



BASIC 프로그램 새로운 디스켓에 INIT HELLO

예

```
5  REM  EX37
6  REM   ********************
7  REM  * YOU CAN'T RESET ! *
8  REM  *                  *
9  REM   ********************
10  REM ..................S.C.H
20  POKE 1012,0
100  HGR
110  HOME
115  VTAB 22
120  PRINT : PRINT " CAN YOU RESET ?"
140  FOR A = 50 TO 250 STEP 50
145  HCOLOR=  INT ( RND (1) * 3) + 1
150  FOR X =  - 20 TO 20 STEP 4
170 B =  INT ( RND (1) * 5) + 1:B = B * 20
180  HPLOT X + A,B TO A,B + 15
190  NEXT : NEXT
200  GOTO 140
```

이 프로그램은 HGR 화면 위에 여러 개의 움직이는 부채꼴 모양을 끝없이 그려나가는 것이다.

만일 다른 사람이 리셋 키이를 눌러, 프로그램을 정지시키거나 리스트를 보려고 하면 즉시 디스크 드라이브가 회전하면서 새로이 부팅이 될 것이다.

응용 38

〈디스크 드라이브 번호〉 $AA68

```
POKE 43624, D
```

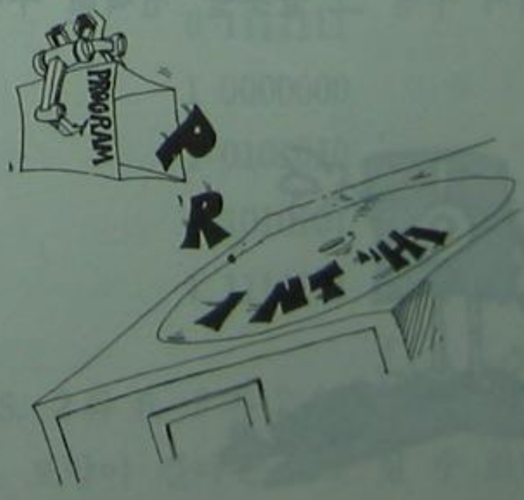
디스크 드라이브 입출력을 D번 드라이브로 바꾼다.

만일 2개의 디스크 드라이브를 사용하고 있는 경우에 1번 드라이브에서 2번 드라이브로 입출력을 바꾸고 싶다면

```
POKE 43624, 2
```

해 주면 된다. 아무 것도 지정하지 않을 경우 43624번지의 값은 1로 되어 있다.

이처럼 사용자가 지정하지 않아도 미리 선택되어 있는 값을 디포울트(default) 값이라고 말한다.



응용 39

〈슬롯 번호〉

$AA6A

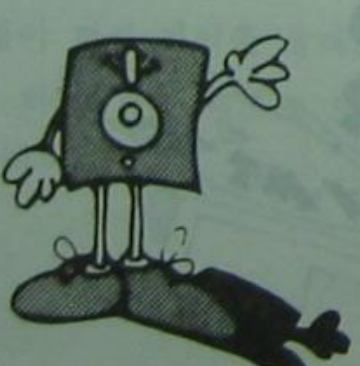
```
POKE 43626, S
```

디스크 드라이브 콘트롤러가 꽂혀 있는 슬롯 S를 드라이브 입출력 슬롯으로 선택한다.

만일 디스크 드라이브를 3개 이상 사용하는 경우라면 슬롯도 2개 이상을 사용해야 한다.

한 슬롯에는 최대 2개의 디스크 드라이브 밖에 연결시킬 수 없기 때문이다.

만일 드라이브 1, 2번이 슬롯 6에, 세번째 드라이브가 슬롯 5에 꽂혀 있고, 현재 드라이브 1, 2번을 사용하여 어떤 결과를 얻은 후에 이것을 다시 세번째 드라이브에 저장시키고 싶다면, POKE 43626, 5하여서 우선 그 슬롯을 선택해 주어야 할 것이다.



응용 40

〈HCOLOR 코우드〉 $E4

```
POKE 228, H
```

H 값에 따라 0번~7번의 HCOLOR 값을 지정해 준다.

APPLE 화면에 컬러로 색을 표시할 때는 그 점을 찍는 위치에 따라 색이 달라지기도 한다. 그 이유는 바로 색깔과 비트 패턴(bit pattern: 화면의 각 비트들이 배열된 상태)이 관계가 있기 때문이다. 0~7번의 HCOLOR 지정에 따른 228번지 내의 비트 패턴을 보자.

HCOLOR	228번지의 2진수	10진수 값
0	0 0000000	0
1	0 0101010	42
2	0 1010101	85
3	0 1111111	127
4	1 0000000	128
5	1 0101010	170
6	1 1010101	213
7	1 1111111	255

0과 4, 1과 5, 2와 6, 3과 7은 MSB(최상위 비트)만 다르고 나머지 7비트는 모양이 같다는 것을 알 수 있다.

흰색으로 쓰이는 3번과 7번은 모든 비트가 1이 되므로 화면의 어느 곳에서 선을 긋더라도 잘 나타나게 되지만, 나머지 컬러들은 세로로 수선(垂線)을 긋는 경우에 컬러 비트가 0인 x축에 오게 되면 화면에는 선이 보이지 않게 될 것이다.

예를 들어 HCOLOR=1이라면 HPLOT 0,10 TO 0, 100은 보이지 않지만 HPLOT 1,10 TO 1, 100은 선이 그어진다(수평으로 가로줄을 그을 때는 상관없다)

예 【256개의 컬러 코우드를 지정】

```
]LOADEX40
]LIST

1  REM  EX40
5  REM  ********************
6  REM  * HCOLOR  CODE     *
7  REM  ********************
9  REM ..................S.C.H
100  HGR
110  FOR C = 0 TO 255
120  POKE 228,C
130  HPLOT 0,0
140  CALL  - 3082
150  NEXT
```

이 프로그램을 RUN시키면 256개의 컬러 비트 패턴에 따라 HGR화면이 칠해져 나갈 것이다.

응용 41

〈Hi-Res 그리는 페이지〉 $E6

```
POKE 230, 32
```

고해상 페이지 1에 그린다.

```
POKE 230, 64
```

고해상 페이지 2에 그린다.

이것은 응용 11에서 했던 POKE-16300, 0이나 POKE-16299, 0과는 전혀 다르다.

응용 11은 어떤 화면을 사용자에게 보여주느냐 하는 것이고 응용 41은 어떤 화면 페이지에 그림을 그리는가 하는 것이다.

따라서 응용 11과 응용 41을 잘 복합시키면, 사용자에게는 페이지 1의 그림을 보여주면서 실제로 내부에서는 페이지 2에다 그림을 그리는 동작을 수행할 수 있는 것이다.

그러므로 페이지 2에 그림을 그려나가는 과정은 화면에 보이지 않다가, 나중에 POKE-16299, 0으로 페이지 2를 보여주게 되면 준비해 두었던 그림이 순식간에 나타나게 된다.

HGR 명령을 내렸을 경우는 그리는 것도 페이지 1이고 보여주는 것도 페이지 1이다. HGR 2명령을 내렸을 경우는 그리는 것도 페이지 2, 보여주는 것도 페이지 2이다.

그러나 POKE 230, X를 사용하면 한 페이지를 보여주면서 또 다른 페이지에는 그림을 그릴 수 있어서, 게임 등의 애니메이션(animation)에 흔히 사용된다.

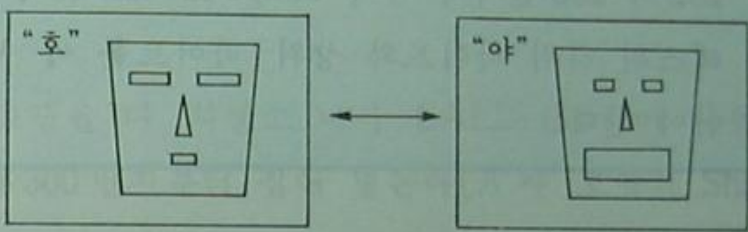
예 【고해상 페이지 1과 페이지 2를 교대로 보이는 법】

```
1   REM  EX41
5   REM  **************************
6   REM  * SHOW PAGE1 & PAGE 2    *
7   REM  **************************
9   REM ..................S.C.H
10   HGR2 : HGR : HCOLOR= 3
15   POKE  - 16302,0
20   GOSUB 130: GOSUB 170
30   POKE 230,64
40   GOSUB 130
50   HPLOT 120,110 TO 230,110 TO 230,135 TO 120,13
     5 TO 120,110
60   POKE  - 16299,0
70   FOR I = 1 TO 200: NEXT
80   POKE  - 16300,0
85   FOR I = 1 TO 600: NEXT
90   GOTO 60
130   HPLOT 100,20 TO 260,20 TO 250,140 TO 110,140
      TO 100,20
140   HPLOT 120,40 TO 160,40 TO 160,50 TO 120,50 TO
      120,40
150   HPLOT 190,40 TO 230,40 TO 230,50 TO 190,50 TO
      190,40
160   HPLOT 175,60 TO 185,100 TO 165,100 TO 175,60

165   RETURN
170   HPLOT 160,110 TO 190,110 TO 190,120 TO 160,1
      20 TO 160,110
200   RETURN
```

- 10행: 두개의 page를 클리어시킨 후 최종적으로 page 1 선택
- 30행: 그리는 것을 page 2로 선택
- 60행: page 2를 보여줌
- 80행: page 1을 보여줌

이 프로그램은 아래와 같은 2개의 그림이 응용 41을 사용하여 페이지 1과 페이지 2에 각각 그려진 후, 응용 11을 사용하여 교대로 페이지 1과 2가 화면에 나타나도록 한 것이다.



페이지 1　　　　페이지 2

이 두 화면이 같은 좌표상에 교대로 나타난다면 마치 이 얼굴이 "야호—", "야호—"를 반복하는 것같은 입모양을 보여주게 될 것이다.

이런 종류의 응용에서 일일히 HGR과 HGR 2를 반복적으로 써준다면 화면이 껌벅거리는 것은 물론이려니와 그 속도 또한 매우 느리게 되어서 실용가치가 없을 것이다.

일단 POKE 230, 64해서 그리게 되면 그 그림은 계속 페이지 2에 보존된다.

주의

HGR이나 HGR 2명령어를 쓰게 되면 Hi-res 그래픽의 page 1과 page 2가 각각 깨끗이 클리어된다.

응용 42

〈Shape Table 시작 번지〉 $E8
$E9

```
POKE 232, LO   ……하위 바이트
POKE 233, HI   ……상위 바이트
```

232와 233번지에 각각 Shape Table의 시작 어드레스의 하위 바이트와 상위 바이트를 써 넣어 주어야 한다.

Shape Table을 사용하려면 BASIC에서 DRAW나 XDRAW를 이용해야 하는데 Apple soft 인터프리터는 Shape Table이 메모리의 어느 위치에 놓여 있는지를 알아야 한다.

APPLE은 232와 233번지에 들어있는 내용을 Shape Table의 시작 어드레스로 간주하게 된다.

그러므로 DRAW나 XDRAW를 사용하기 이전에 두 개의 POKE문을 사용하여 Table의 위치를 알려주어야 하는 것이다.

예 **【굴러가는 A자】**

```
5  REM  EX42
6  REM  **************************
7  REM  * USE OF  SHAPE-TABLE    *
8  REM  **************************
9  REM  ...................S.C.H
20  POKE 232,0: POKE 233,3
```

```
30  HGR : HCOLOR= 3
35  HPLOT 0,20 TO 240,159  비탈길을 그린다
40  FOR X = 10 TO 255
50  SCALE= 2
60  ROT= X
70  DRAW 1 AT X,X / 1.7     A자를 그린다
80  XDRAW 1 AT X,X / 1.7    A자를 지운다
90  NEXT X
100  GOTO 40
```

이 프로그램을 다 쳐넣고 나서 CALL-151 하여 다음의 Shape Table 을 $300 번지부터 집어 넣는다(A 자 모양의 Shape Table 임).

```
*300: 01 00 04 00 24 24 05 28 28
      AA 36 36 07 D8 28 2D 00
```

다시 CTRL-C 하여 BASIC 으로 돌아온 다음 RUN 하여 보자. A 자가 비탈길에서 굴러 내려가고 있게 될 것이다.



응용43

〈DOS 명령어의 조작 1〉 $ A884~ $ A887

LOAD 명령어를 바꾸기

```
POKE 43144, A
POKE 43145, B
POKE 43146, C
POKE 43147, D
```

(A, B, C, D는 문자의 ASCII 코우드)

이번에는 DOS 명령어를 자유자재로 바꾸는 방법을 알아보자.

예를 들어 "CATALOG"라는 명령을 "SHOW ME"로 바꾼다면, 그 바뀐 DOS를 가지고 아무리 다른 사람이 CATALOG를 보려고 시도해 보았자 SYNTAX ERROR 메시지만 받게 될 것이다.

이처럼 DOS 명령어를 바꾸는 것은 당신의 프로그램에 보호 장치(protect)를 만드는 한 방법이기도 하다.

위의 A, B, C, D는 바뀔 문자의 ASCII 코우드를 나타내며 이 경우 바뀔 명령어의 글자 수는 최대 4자(LOAD가 4자이므로), 최소 1자까지 조절할 수 있다.

예 【LOAD 를 → BEER 로】

```
]LIST

1   REM   EX43
5   REM   *************************
6   REM   *    CHANGE 'LOAD'      *
7   REM   *************************
9   REM   ..................S.C.HAHN
10   POKE 43144,66
20   POKE 43145,69
30   POKE 43146,69
40   POKE 43147,82 + 128
50   REM      ....MAIN PRGRAM FOLLOW..
```

여기서 66 은 B 의 아스키 코우드, 69 는 E 의 아스키 코우드이며 82 는 R 의 아스키 코우드이다. 그런데 마지막 글자인 R 에는 왜 **128 을 더했을까?** 이것은 '이곳이 **명령어의 끝 글자**이다' 하는 것을 컴퓨터에게 알려 주기 위하여 **마지막 글자의 MSB(비트 7)를 1 로 세트시켜** 주는 방법을 쓰기 때문이다.

이제 위의 프로그램을 RUN 시킨 다음,

```
BEER HELLO
```

해 보자. 디스크가 돌아가며 HELLO 프로그램이 로우드된다. 남들이 아무리 LOAD 를 해 봤자 작동하지 않을 것이다.

응용44

〈DOS 명령어의 조작 2〉

$ A890～$ A892

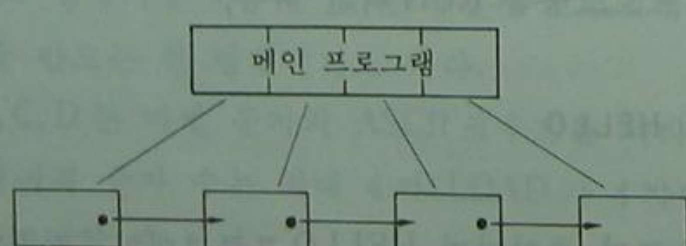
RUN을 바꾸려면

```
POKE 43152, A
POKE 43153, B
POKE 43154, C
```

(A, B, C는 문자의 ASCII 코우드)

DOS명령어 중 하나인 RUN filename을 다른 명령어로 바꾸어 버린다면, 다른 사람은 그 DOS상에서 당신이 짠 프로그램을 수행시키는데 상당히 애를 먹게 된다.

물론 이러한 방법 한 가지로는 **프로그램의 보호장치**로 불충분하지만, 앞에 나온 'LIST를 못보게 하는 방법'에다 덧붙여서, 메인 프로그램을 여러 개로 나누어 앞의 프로그램 맨 뒷부분에 그 다음 프로그램을 RUN시키는 명령어를 넣도록 한다면 꽤 효과적인 방법이 될 수도 있을 것이다.



화살표의 •부분에서 변경된 RUN명령을 내림

예 【RUN 을 → GO 로】

```
]LOADEX44
]LIST

1   REM  EX44
5   REM  *************************
6   REM  *  CHANGE 'RUN'         *
7   REM  *************************
9   REM  ..................S.C.H
10   POKE 43152,71
20   POKE 43153,79 + 128
30   POKE 43154,64
40   PRINT  CHR$ (4)"GO HELLO"
```

이 프로그램을 일단 RUN 시키고 나면 DOS 명령어인 RUN filename 은 GO filename 을 해야만 작동하게 된다. 이 프로그램의 마지막 줄에서 GO HELLO 는 디스크 속에 들어있는 HELLO 란 프로그램을 찾아서 자동으로 수행시킬 것이다.

10 번 문장에서는 G 에 해당하는 코우드 71 을 썼고 20 번에서는 O 에 해당하는 79 에다 명령어의 마지막임을 알리는 128 을 더해 주었고, 30 번에서는 빈 칸에 대한 아스키 코우드 64 를 써 주었다.

응용45

〈DOS 명령어의 조작 3〉 $ A8D2~ $ A8D8

CATALOG 명령어 바꾸기

```
POKE 43218, A
POKE 43219, B
POKE 43220, C
POKE 43221, D
POKE 43222, E
POKE 43223, F
POKE 43224, G
```

(A~G는 원하는 문자의 ASCII 코우드)

CATALOG 명령어를 다른 명령어로 바꾸어 당신 자신만의 DOS로 INIT(이니셜라이즈)시켜 놓는다면, 이것은 당신 혼자만이 그 암호를 알고 있는 DOS가 될 것이다.

일단 보통 DOS를 부팅한 후 지금 하고 있는 명령어 변환 작업들을 해 놓은 뒤, 새로운 HELLO 프로그램으로 INIT(이니셜라이즈)시켜 놓으면 된다.

다음 예는 CATALOG란 명령어를 SHOW!로 바꾸는 것이다.

예 【CATALOG를 → SHOW!로】

```
]LOADEX45
]LIST

1   REM  EX45
5   REM  **************************
6   REM  *    CHANGE  'CATALOG'  *
7   REM  **************************
9   REM  .................S.C.H
10   POKE 43218,83
20   POKE 43219,72
30   POKE 43220,79
40   POKE 43221,87
50   POKE 43222,33 + 128
60   POKE 43223,32 + 128: REM " "
70   POKE 43224,32 + 128: REM " "
80   PRINT  CHR$ (4)"SHOW!"
```

이 프로그램을 RUN시키고 나면 자동으로 카탈로그를 보여주게 되며, 이후로 SHOW!라고 키이를 누를 때마다 카탈로그를 보여 줄 것이다.



응용 46

〈DOS 명령어의 조작 4〉

$ A8F9
~
$ A8FC

BLOAD 명령어를 바꾸려면

```
POKE 43257, A
POKE 43258, B
POKE 43259, C
POKE 43260, D
```

(A, B, C, D 는 원하는 문자의 ASCII 코우드)

예 **【BLOAD 를 → 〉〉〉로】**

```
]LOADEX46
]LIST

1  REM  EX46
5  REM  **************************
6  REM  *    CHANGE  'BLOAD'     *
7  REM  **************************
9  REM  ..................S.C.H
10  POKE 43257,62
20  POKE 43258,62
30  POKE 43259,62 + 128
40  POKE 43260,32 + 128
50  POKE 43261,32 + 128
```

이 프로그램을 RUN 시킨 후

〉〉〉 화일 이름

하면 그 기계어 프로그램이 로우드될 것이다.

응용 47 〈DOS 명령어의 조작 5〉

BRUN 을 바꾸기 $A8FE～$A901

```
POKE 43262.A
POKE 43263.B
POKE 43264.C
POKE 43265.D
```

(A, B, C, D는 원하는 문자의 ASCII 코우드)

예 **【BRUN 을→LOVE 로】**

```
]LIST

1  REM  EX47
5  REM  **************************
6  REM  *  'LOVE' IS  'BRUN'     *
7  REM  **************************
9  REM  ....................S.C.H
10  POKE 43262,76
20  POKE 43263,79
30  POKE 43264,86
40  POKE 43265,69 + 128
```

이 프로그램을 RUN 시킨 후 원하는 기계어 프로그램 이름(binary file name)을 다음과 같이 실행시킬 수 있다.

```
LOVE  화일 이름
```

응용 48

〈BASIC 프로그램의 시작 위치〉 $67, $68

BASIC 프로그램을 LOC라는 번지부터 로우드(LOAD)시키려면

```
POKE 103, LOC-INT(LOC/256)*256
POKE 104, INT(LOC/256)
POKE LOC-1,0
```

(103, 104번지에는 BASIC 프로그램의 시작 어드레스가 들어 있다)

【원하는 메모리 위치에 프로그램을 싣고 싶으면】

Apple에는 BASIC으로 짠 프로그램을 메모리 내에 자유자재로 위치시킬 수 있는 방법이 있다.

제로 페이지(zero page)의 $67, $68번지에는 BASIC 프로그램의 시작 어드레스가 들어 있는데, 평상시 $67번지에는 01이 담겨 있고 $68번지에는 08이 담겨져 있어서 결과적으로 $0801번지, 즉 십진수로 2049번지부터 프로그램이 위치하게 되어 있다. 그러나 당신이 어떤 필요로 인해서 그 위치를 바꾸고 싶다면 $67, $68번지에 들어 있는 내용을 원하는 어드레스로 바꾸어 주기만 하면 된다.

이러한 필요는 메모리 안에 두 개의 독립된 프로그램을 넣고 싶을 때(대개 이 중 하나는 기계어 서브루우틴인 경우가 많다) 생겨난다.

예를 들어 BASIC 프로그램을 $4001 번지부터 넣고 싶다면

$67 에 ──→ 01 을

$68 에 ──→ 40 을 집어 넣어주면

그다음부터 로우드(LOAD)되는 프로그램이나 키이보오드를 통해 입력되는 프로그램은 $4001 번지부터 자리잡게 될 것이다.

그리고 프로그램이 시작되는 바로 앞 번지에는 프로그램의 시작임을 알리는 0이 들어가 있어서 평상시에는 $800 번지에 0이 있고, 위에 든 예에서라면 $4000 번지에 0을 넣어두어야 한다.

이러한 과정들을 위에서 설명한 3개의 POKE 문을 통해 실현할 수 있게 된다.

【극비 프로그램을 숨겨두는 비밀구좌】

그러면 지금부터 2개의 독립된 BASIC 프로그램을 각각 다른 영역의 메모리에 집어 넣어보자(한 개의 진짜 프로그램, 또 하나의 그것을 숨기기 위한 위장용 프로그램).

당신의 중요한 프로그램을 작성하여 디스켓에 저장시킨 후 다시 다음과 같은 프로그램을 짠다.

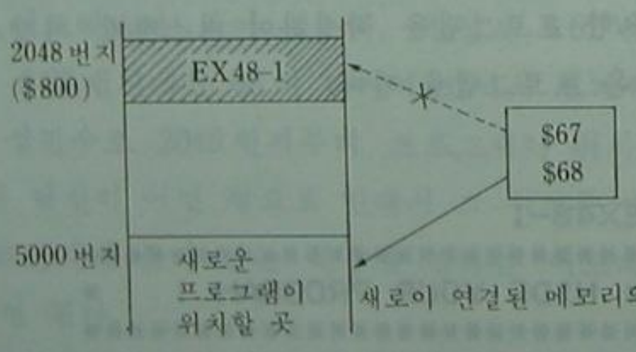
```
1   REM   EX48-1
5   REM   **************************
6   REM   * HIDE YOUR PROGRAM !    *
7   REM   **************************
9   REM ..................S.C.H
10   HOME
20   FOR I = 1 TO 24 * 40: PRINT "?";: NEXT
```

```
30  VTAB 12: HTAB 15
40  INPUT "  SECRETE CODE: ";CODE
50  POKE 103,CODE -  INT (CODE / 256) * 256
60  POKE 104, INT (CODE / 256)
70  POKE CODE - 1,0
```

이 때 $67번지에는 01이, $68번지에는 08이 들어가 있다. 그러므로 현재의 이 프로그램은 메모리의 $801(십진 2049)번지부터 들어있게 된다.

이제 실제로 숨겨두고 싶은 당신의 프로그램이 들어갈 자리를 결정하자. 여기선 임의로 십진 5001번지로 선택했다. 이것이 바로 당신만이 알고 있는 암호이다. 위의 프로그램을 RUN하고 SECRETE CODE:에 5001번을 쳐 넣는다.

이제 LIST를 쳐보자. 아니? 아무것도 나타나지 않는다. 방금 RUN시켰던 프로그램은 어디로 없어진 것일까? 그게 아니다. 다음 그림을 보도록 하자.



메모리의 2049번지로 연결되었던 포인터가 5001번지로 다시 연결됨으로써 2049번지부터 있던 프로그램은 지워지지는 않고 남

아 있지만 사실상 사용할 수 없는 프로그램이 되어버린 것이다.

이제 당신의 그토록 숨기고 싶어하는 극비의 BASIC 프로그램을 LOAD 시킨다. 디스켓에 저장시켜 두었던 프로그램을 EX48-MAIN 이라고 한다면,

```
LOAD EX48-MAIN
```

하면 당신이 숨기고자 하는 이 프로그램은 EX48-1 프로그램을 건드리지 않고 5001 번지부터 LOAD 되어 자리잡게 될 것이다.

LIST 시켜서 확인하여 보자. 방금 로우드시킨 프로그램이 들어가 있다. 이제부터가 중요한 대목이다. CALL-151 하여 $67 번지와 $68 번지를 보게되면

```
0067-89
0068-13
```

이것은 $1389 번지, 즉 십진 5001 번지이다. 이제 이것을 다시 $801 번지로 고쳐보자.

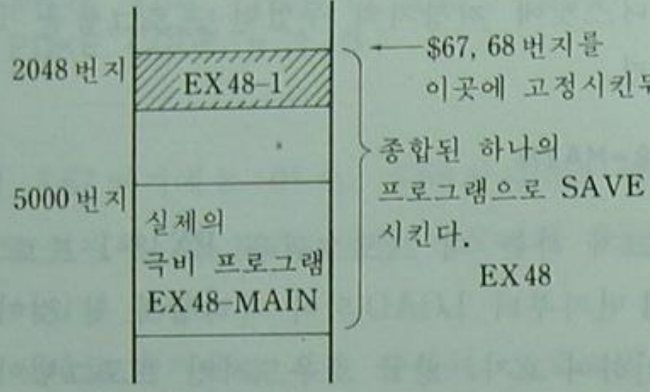
```
*67:01 08
```

하고 리턴 키이를 누르면 된다.

다시 CTRL-C 를 누르고 리턴을 눌러서 BASIC 으로 돌아온 후 LIST 시켜보면…… 아! EX48-1 프로그램이 다시 살아나왔다!

현재 메모리에는 EX48-1 과 EX48-MAIN 의 두 개의 프로그램이 들어있지만 $67, 68 번지에 들어있는 BASIC 프로그램의 포인터가 $801 번지를 가리키므로 EX48-1 만이 제 구실을 하고 있는 것이다. 이처럼 두 개의 프로그램이 메모리 내에 함께 들어 있는 상태에서 이것을 하나의 프로그램으로 간주하여 새로운 이름으로

SAVE 시켜 보자.




```
SAVE EX48
```

SAVE 시킨 후에 CATALOG 를 보면 이 EX48 프로그램의 길이가 (EX48-1 의 길이)+(EX48-MAIN 의 길이)보다 훨씬 긴 것을 발견하게 될 것이다. 이것은 두 프로그램 사이의 공백까지도 프로그램의 일부로 간주되어서 SAVE 되었기 때문이다.

자, 이제 모든 준비는 다 된 셈이다. 새롭게 탄생된 EX48 프로그램을 RUN 시키자. 그러면 **비밀번호를 물어보는 EX48-1 의 프로그램이 수행**된다. 여기에 **당신만이 알고 있는 비밀번호,** 즉 5001 번을 **키이로 눌러준 후** 다시 RUN 을 시키게 되면 깊숙히 숨겨두었던 EX48-MAIN 프로그램이 수행될 것이다.

암호 코우드를 묻는 과정에서 5001 번 이외에 다른 번호를 답하게 되면 절대로 숨겨져 있는 프로그램을 RUN 시킬 수 없다. **스위스 은행의 비밀구좌처럼** 당신도 **당신만이 RUN 시킬 수 있는 프로그램상의 비밀구좌**를 만들 수 있게 된 것이다.

응용 49

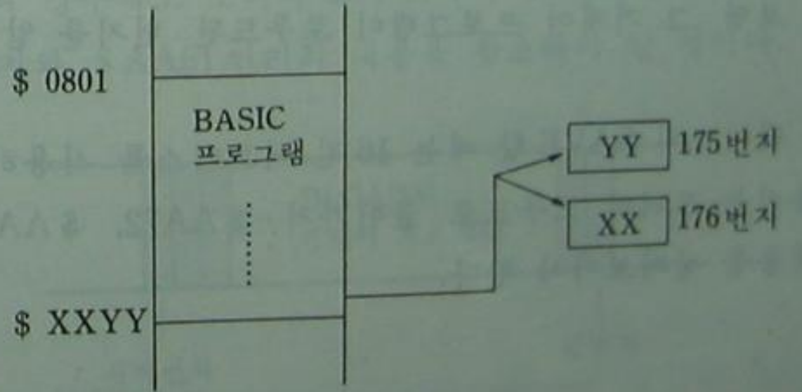
〈BASIC 프로그램의 끝 위치〉 $AF.B0

```
PEEK(175)+PEEK(176)*256
```

현재 메모리에 들어있는 Applesoft 프로그램의 끝 위치가 몇 번지인가를 알려준다.

당신이 BASIC 프로그램을 짜 나가는 동안 쓸 수 있는 메모리는 점점 줄어든다. 또한 메모리의 윗부분에 기계어 서브루우틴을 위치시키려고 할 때에도 BASIC 프로그램이나 프로그램 변수가 들어가는 영역과 겹치지 않도록 해야 한다.

이런 경우 지금 몇 번지까지 BASIC 프로그램이 들어가 있는가를 살펴보기 위해서는 위의 PEEK 문을 프린트시키면 될 것이다.



응용 50

〈BLOAD된 기계어 프로그램의 시작번지〉

$ AA72
$ AA73

```
PEEK(43634)+PEEK(43635)*256
```

가장 최근에 BLOAD된 binary 프로그램의 시작 위치를 담고 있다.

재미있는 기계어 게임 프로그램이 있는데 이것은 당신의 친구에게도 복사해주고 싶다. 그런데 그 친구는 아직 디스크 드라이브를 장만하지 못했다고 하자. 이런 경우 어쩔 수 없이 카세트 테이프에 복사해 두는 수 밖에 없는데, **기계어 프로그램의 시작번지와 끝번지를 모르면 테이프에 SAVE할 수 없다.** 그러므로 기계어 프로그램을 디스켓에서 BLOAD시킨 후 위의 PEEK문을 프린트해 보면 그 기계어 프로그램이 로우드된 번지를 알 수 있다.

그러나 테이프에 SAVE할 때는 16진 어드레스를 사용하므로, 실제 사용시는 모니터 모우드로 들어가서 $ AA72, $ AA73 두 번지의 내용을 살펴보아야 한다.

응용 51

〈BLOAD된 기계어 프로그램의 길이〉

$ AA60
$ AA61

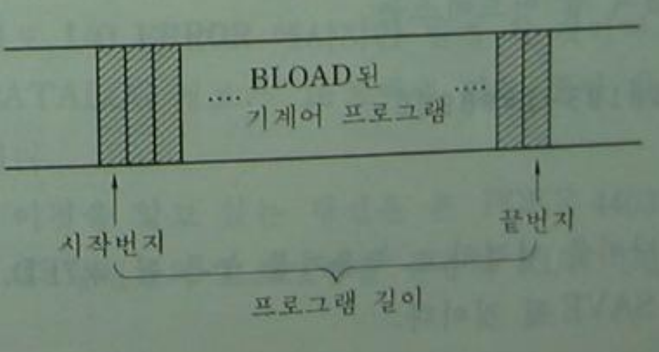
```
PEEK(43616)+PEEK(43617)*256
```

가장 최근에 BLOAD된 프로그램의 길이(byte 수)를 담고 있다.

위의 PEEK문을 프린트해 보면, 즉각 로우드된 기계어 프로그램의 길이를 알아볼 수 있다(byte 수로).

또한 응용 50에서 다루었던 문제, 즉 디스켓에 있는 기계어 프로그램을 테이프에 복사하는 과정에서 그 기계어 프로그램의 끝번지 수를 알아야 하는데, 이것은 **시작번지+프로그램 길이－1**로 계산해 낼 수 있다.

위의 경우에도 모니터 모우드에서 16진수를 사용하므로 $AA60번지와 $AA61번지의 내용을 참조해야 할 것이다.



【디스켓의 기계어 프로그램을 카세트에 옮기기】

그러면 실제로 예를 통하여 응용 50, 51을 활용해 보자. 시중에서 구하기 쉬운 기계어 게임 프로그램인 BEER RUN을 예로 들었다.

디스켓을 부팅시킨 후,

```
BLOAD BEER RUN
```

하면 기계어 프로그램이 메모리에 로우드된다. 커어서가 나오면 CALL-151 하여 모니터 모우드로 들어간 후 *AA72. AA73을 치고 리턴을 누르면

```
AA72-FD 07
```

처럼 나올 것이다. 즉 프로그램의 시작 어드레스가 $7FD라는 뜻이다. 다시 *AA60. AA61을 치고 리턴을 누르면

```
AA60-03 81
```

처럼 나오므로 이 프로그램의 길이는 16진수로 8103바이트이다. 따라서 프로그램의 끝 어드레스는

```
$7FD+$8103-1=$8900
```

이 된다.

이제 카세트 단자를 연결하고 녹음기를 누른 뒤 *7FD. 8900W 하면 카세트에 SAVE될 것이다.

응용52

〈CATALOG 트랙 번호〉 $ AC01

```
PEEK(44033)
```

44033 번지를 PEEK 하면 CATALOG 가 들어있는 디스켓의 트랙 번호가 나온다.

일반적으로 위의 PEEK 문을 프린트시키면 17이 나오는데, Apple 의 DOS 에서는 트랙 17에 CATALOG 에 관한 정보를 담기 때문이다.

만일 당신이 그리팅 프로그램(디스켓을 INIT 시킬 때 사용하는 것으로, 디스켓이 부팅되면서 처음으로 수행되는 프로그램) 속에 POKE 44033, 16 등으로 CATALOG 트랙 번호를 조작해 놓으면, **영문을 모르는 다른 사람들은** 이 그리팅 프로그램이 일단 수행된 후에는 아무리 CATALOG 를 치거나 프로그램을 RUN 해 보려고 해도 **I/O ERROR 메시지만** 받게 될 것이다. 디스크 드라이브는 CATALOG 정보가 16 트랙에 있는 줄만 알고 계속 헤메기 때문이다.

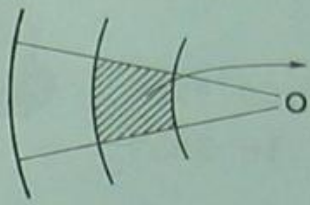
그러나 이것을 알고 있는 당신은 곧 POKE 44033, 17 로 정상 환원시킨 후 당신만의 프로그램을 유유히 RUN 시킬 수도 있다.

응용 53

〈디스켓의 한 섹터를 읽어들이기〉

다음 프로그램을 수행시키면 디스켓에 있는 어떤 트랙(track), 몇째 섹터(sector)의 256 바이트라도 읽어낼 수 있다.

```
1 REM  EX53
5 REM  **************************
6 REM  * READ T-TRACK,S-SECTOR *
7 REM  **************************
9 REM
10  HOME
20  INPUT "WHICH TRACK:";T
30  INPUT "WHICH SECTOR:";S
40  POKE 768,32: POKE 769,227: POKE 770,3: POKE
    771,76: POKE 772,217: POKE 773,3
50  POKE 47083,0: POKE 47088,0: POKE 47089,1
    50: POKE 47090,0: POKE 47091,0: POKE 47
    092,1
60  POKE 47084,T: POKE 47085,S
70  CALL 768
75 MEM = 38400
80  FOR I = MEM TO MEM + 255
90  PRINT "BYTE:";I - MEM,"VALUE->"; PEEK (I
    )
95 K = K + 1: IF K = 23 THEN K = 0: GET A$
100  NEXT
```



이 정보들은 카피 프로텍트(copy protect) 등을 연구하는데 중요하게 이용되기도 한다.

예 트랙 1, 섹터 1

```
]RUN
WHICH TRACK:1
WHICH SECTOR:1
BYTE:0          VALUE->38
BYTE:1          VALUE->69
BYTE:2          VALUE->96
BYTE:3          VALUE->40
BYTE:4          VALUE->32
BYTE:5          VALUE->164
BYTE:6          VALUE->161
BYTE:7          VALUE->240
BYTE:8          VALUE->197
BYTE:9          VALUE->56
BYTE:10         VALUE->233
BYTE:11         VALUE->176
BYTE:12         VALUE->48
BYTE:13         VALUE->238
BYTE:14         VALUE->201
BYTE:15         VALUE->10
BYTE:16         VALUE->144
BYTE:17         VALUE->8
```

응용54

〈역어셈블러; Disassembler〉 $FE61

```
POKE 58, LO
POKE 59, HI
CALL-415
```

58, 59번지에 그 시작 어드레스가 들어있는 기계어 프로그램을 어셈블리 어 프로그램 형태로 바꾸어 화면에 리스트시킨다(한번 CALL-415할 때마다 20줄씩 리스트됨). 리스트시킨 후에도 계속 BASIC 모우드(])로 남아 있게 된다.

역어셈블러 기능은 사용자가 POKE문을 사용하여 기계어 루우틴을 짠 후, 이것이 제대로 들어가 있는지 확인해 보고 싶을 때 BASIC 상태에서 직접 사용할 수 있다.

예를 들어 4C 3C D4의 기계어는 JMP $D43C로 바뀌어서 리스트되는 것이다.

메모리 어드레스 $0A번지부터 역어셈블된 내용을 리스트시키고 싶다면 다음과 같이 하면 된다.

```
POKE 58,10  } $000A 번지
POKE 59,0   }
CALL-415
```

응용55

〈2 바이트의 어드레스를 POKE 해 넣기〉

```
POKE LO, ADR-INT(ADR/256)*256
POKE HI, INT(ADR/256)
```

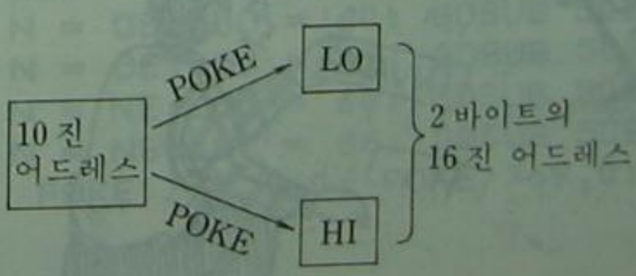
ADR : 0~65536 의 십진 어드레스

LO : 어드레스 하위 바이트가 들어갈 번지

HI : 어드레스 상위 바이트가 들어갈 번지

Apple 이 사용하는 어드레스를 메모리에 넣어두려면 2 개의 장소가 필요하게 된다. 그런데 6502 는 일반적으로 어드레스의 하위 바이트부터 저장, 재생시키므로 연속된 2 개의 장소 중 앞에는 하위 바이트, 뒤에는 상위 바이트를 담게 된다.

따라서 위의 두 POKE 문은 **10 진 어드레스를 16 진 어드레스로 변환시킴**과 동시에 그것을 **두 바이트로 나누어** 저장을 하는 구실을 한다. 상위 바이트의 자리 단위는 하위 바이트의 256 배임을 생각한다면 POKE 문에서 사용되는 수식을 이해할 수 있을 것이다.



응용 56

〈DELETE된 화일 이름의 재생〉

```
POKE 44505, 234
POKE 44506, 234
```

한 후

```
CATALOG
```

일단 'DELETE 화일 이름' 하게 되면 그 화일 이름은 다시는 CATALOG에 나타나지 않는다.

그러나 때에 따라서는 자신이 무슨 프로그램을 DELETE시켰는지를 기억 못할 경우도 있으므로, **다시 DELETE시켰던 화일의 이름을 보고 싶은 경우**가 생긴다.

이럴 때 위의 두 POKE문을 수행시킨 후 CATALOG를 시키면 DELETE되었던 화일 이름이 되살아난다.



응용57 〈메모리 블럭의 이동〉

다음 프로그램의 5000번 이하의 서브루우틴은 메모리 어드레스 OS~OE에 걸쳐 있는 내용(프로그램 혹은 데이터)을 NS~NE의 메모리 영역으로 전송해 준다.

```
]LIST

0  REM  EX57
1  REM  **************************
2  REM  *  MEMORY MOVEMENT       *
3  REM  **************************
10  REM   OS--OLD START.
11  REM   OE--OLD END.
12  REM   NS--NEW START.
13  REM   NE--NEW END.
100  HGR : HCOLOR= 3
110  HPLOT 0,0 TO 279,0 TO 279,3 TO 0,3 TO 0
     ,0
120  INPUT "OS,OE =";OS,OE
130  INPUT "NS,NE =";NS,NE
200  GOSUB 5000
220  END
5000 N = OS:LOC = 60: GOSUB 5020
5001 N = OE:LOC = 62: GOSUB 5020
5002 N = NE:LOC = 64: GOSUB 5020
5003 N = NS:LOC = 66: GOSUB 5020
5010  POKE 768,160: POKE 769,0: POKE 770,76:
```

```
     POKE 771,44: POKE 772,254: CALL 768:
     RETURN
5020 POKE LOC,N - INT (N / 256) * 256: POKE
    LOC + 1, INT (N / 256): RETURN
```

여기서 OS와 OE는 각각 옮기기 전의 메모리 어드레스의 시작 번지와 끝번지이고 NS와 NE는 옮겨갈 곳의 시작 번지와 끝 번지이다.

앞의 프로그램을 RUN시키면 HGR화면상에 긴 막대가 그려진 후 두 개의 INPUT문장에 의해 옮겨야 할 메모리 블럭과 옮겨져 갈 위치의 어드레스들을 입력시키고 나면, 즉시 HGR화면의 내용이 옮겨간 것을 볼 수 있다.

예를 들어

```
OS, OE=8192, 12000
NS, NE=12092, 16000
```

로 데이터를 넣어주면 화면에 그려졌던 막대의 왼쪽 반 정도가 화면의 아랫쪽에 옮겨져 오게 된다. 이런 현상은 HGR화면이 연속적으로 메모리 할당되어 있지 않기 때문에 생기는 결과이지만, 메모리 블럭이 전송되는 것을 시각적으로 쉽게 느낄 수 있게 하기 위하여 HGR화면을 사용하였다.

NS, NE를 여러 가지로 바꾸어 보며 RUN시켜보자.

응용 58

〈음향 효과 서브루우틴〉

$302 번지부터 다음의 기계어 서브루우틴을 넣고 BASIC 모우드로 돌아와서 $300 번지와 $301 번지에 POKE 문으로 데이터를 넣어준 후 이 서브루우틴을 CALL 한다.

```
*302L

0302-    A2 33          LDX   #$33
0304-    AD 00 03       LDA   $0300
0307-    20 A8 FC       JSR   $FCA8
030A-    AD 30 C0       LDA   $C030 … 스피커에
030D-    AD 01 03       LDA   $0301    신호
0310-    20 A8 FC       JSR   $FCA8
0313-    AD 30 C0       LDA   $C030 … 스피커에
0316-    CA             DEX            신호
0317-    E0 00          CPX   #$00
0319-    D0 E9          BNE   $0304
031B-    60             RTS
```

위의 기계어 서브루우틴은 **두 개의 딜레이 상수**(delay constant) **에 의해서 스피커에 신호를 주는 주기를 결정**하게 되므로, BASIC 에서 이들을 조정함에 따라서 여러 가지 음색과 음 높이를 소프트웨어 상으로 만들어 볼 수 있다.

기본적인 BASIC 루우틴은 다음 구조를 갖는다.

```
POKE 768, D1:POKE 769, D2
CALL 770
```

예 1 【귀뚜라미 소리내기】

```
1  REM  EX58.1
5  REM  ***************************
6  REM  *   INSECT SOUND          *
7  REM  ***************************
10  REM  ..................S.C.H
90  FOR X = 1 TO 6
100  FOR I = 1 TO 10
110 D1 = 5:D2 = 4
115  GOSUB 1000
120  NEXT
130  FOR P = 1 TO 200
140  NEXT P
150  NEXT X
200  END
1000  REM <CALL MACHINE-ROUTINE>
1010  POKE 768,D1
1020  POKE 769,D2
1030  CALL 770
1040  RETURN
```

여기서는 100~120 까지의 루우프가 귀뚜라미 소리를 내는 루우틴이고 130~140 문장은 중간 중간에 소리를 멈추기 위한 딜레이 루우프이다.

1000번 이후는 기계어 서브루우틴을 부르는 BASIC 루우틴으

로써 자기 자신도 주프로그램의 서브루우틴으로 사용되고 있다.



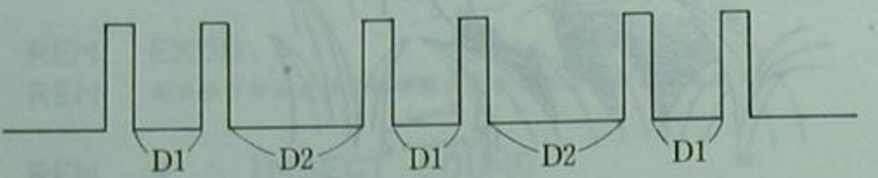
예 2 【전화기 전자벨 소리내기】

```
LIST

1  REM  EX58.2
5  REM  ****************************
6  REM  *    TELEPHONE SOUND       *
7  REM  ****************************
10  REM  ...................S.C.H
90  FOR X = 1 TO 5
100  FOR I = 1 TO 15
110 D1 = 7:D2 = 20
115  GOSUB 1000
120  NEXT
150  FOR P = 1 TO 750: NEXT P
190  NEXT X
200  END
1000  REM <CALL MACHINE-ROUTINE>
1010  POKE 768,D1
1020  POKE 769,D2
1030  CALL 770
1040  RETURN
```

이 프로그램도 앞의 예제와 그 구성 방법은 동일하다.

단지 110번에서의 D1과 D2를 좀더 큰 값으로 바꾸어 줌으로써 주기를 길게 해서 음색깔이 달라지도록 한 것이다.

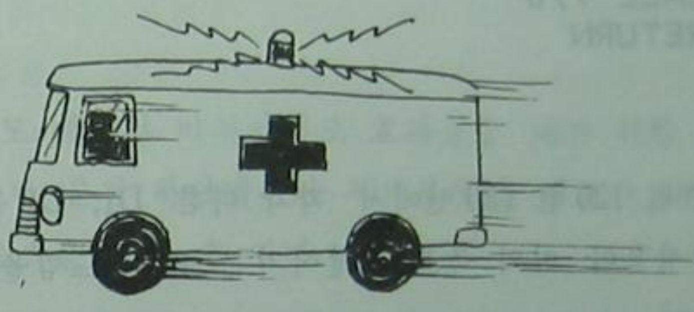


예 3 【앰블런스 소리내기】

```
1  REM  EX58.3
5  REM  ***************************
6  REM  *    AMBULANCE SIREN      *
7  REM  ***************************
10  REM  ...................S.C.H
90  FOR X = 1 TO 7
100  FOR I = 1 TO 2
110 D1 = 3:D2 = 30
115  GOSUB 1000
120  NEXT
125  FOR I = 1 TO 2
130 D1 = 15:D2 = 30: GOSUB 1000
140  NEXT
190  NEXT X
200  END
1000  REM <CALL MACHINE-ROUTINE>
1010  POKE 768,D1
1020  POKE 769,D2
1030  CALL 770
1040  RETURN
```

이 프로그램은 두 개의 음색을 섞어서 교대로 소리나게 하여, 마치 앰뷸런스의 사이렌 소리와 같은 효과를 내도록 해본 것이다.

100~120이 한 개의 음색깔을, 125~140이 또다른 음색깔을 만들어 내며 전체적으로 7번 반복 수행하도록 되어 있다.



예 4 【환상적인 전자 폭포수 1】

```
]LIST

1   REM   EX58.4
5   REM   **************************
6   REM   *  ELECTRONIC WATER FALL*
7   REM   **************************
10   REM  ....................S.C.H
90   FOR X = 1 TO 30
100 D1 = X:D2 = 31 - X
110  GOSUB 1000
120 D1 = 3:D2 =  INT (X / 2) + 1:
      GOSUB 1000
```

```
130 D1 = 5:D2 = 3: GOSUB 1000
150  NEXT X
200  END
1000  REM <CALL MACHINE-ROUTINE>
1010  POKE 768,D1
1020  POKE 769,D2
1030  CALL 770
1040  RETURN
```

여기서는 100번 120번 130번에서 각각 다른 D1, D2를 사용함으로써 그 합성음이 마치 폭포가 떨어지는 듯한 느낌을 줄 수 있도록 해 보았다.

120번에서 INT(X/2)를 쓴 것은 X값이 너무 크기 때문에 줄이기 위한 것이고 1을 더해준 것은 D2의 값이 0이 되지 않도록 하기 위해서이다.

예 5 【환상적인 전자 폭포수 2】

```
1  REM  EX58.5
5  REM  ***************************
6  REM  *  ELECTRONIC WATER FALL*
7  REM  ***************************
10  REM  ...................S.C.H
80  FOR I = 1 TO 4
90  FOR X = 1 TO 20
100 D1 =  INT (X / 2) + 1:D2 = X:
     GOSUB 1000
150  NEXT X
190  NEXT I
```

```
200  END
1000  REM <CALL MACHINE-ROUTINE>
1010  POKE 768,D1
1020  POKE 769,D2
1030  CALL 770
1040  RETURN
```

이것도 예 4와 마찬가지로 효과음을 내기 위한 것이다. 이 경우 100번 문장 하나에서만 서브루우틴 점프가 일어나므로 더 깨끗한 음색을 얻을 수 있다.



* 맺 음 말 *

처음 이 책을 쓰리라 마음 먹었을 때는 Apple 활용의 극대화를 꾀한다는 목표아래 펜을 들었지만, 원고를 끝내는 지금의 심정은 오직 송구스러울 따름이다. PEEK, POKE에 대한 단행본을 낸다는 것이 쉬운 일이 아니었음을 늦게서야 깨닫게 되었다.

독자 여러분께 백배 사죄하고, 더 유익한 책을 쓰는데 시금석이라도 될 수 있다면 천만 다행이리라.

끝까지 읽어주신 독자 여러분께 감사하며, 뜨거운 격려 전화 주셨던 현대 전자의 金형에게도 서면 인사 드린다.

부록 A

용어 해설

① **반도체 칩 :** 예전에는 진공관이나 트랜지스터를 사용했으나 요즈음은 여러 개의 반도체를 하나의 패키지(package)안에 집어넣어서 단일 칩(chip, 조각, 부품 하나)으로 된 전자회로를 상품화하고 있다. 이것은 몇 가지의 규격이 통일되어 있으므로 칩에 붙어있는 핀들도 규정에 따라 배열이 일정하다.

구리같은 것은 언제나 전기가 통하는 도체이고, 사기같은 재료는 항상 전기가 통하지 않는 부도체이다. 이에 비하여 반도체란 가해지는 전압, 전류 및 외부 조건에 의하여 어떤 때는 전기가



그림 1 반도체 칩

통하지 않고 또 어떤 때는 전기가 통하는 물질로서, 순도가 높은 원소로 이루어진 물질에 극소량의 불순물 원소를 집어 넣어 만들어진다.

② **IC(집적회로)** : IC는 여러 개의 전자회로를 하나의 패키지로 집적하여 만들었다는 Integrated Circuit의 약자로서 여러 회로가 들어있는 반도체 칩이다. 한 개의 IC가 어떤 독특한 기능을 하는 회로를 내장하고 있으므로, 그 IC가 고장났을 때는 소켓에 IC만 갈아끼우면 되도록 되어있다.

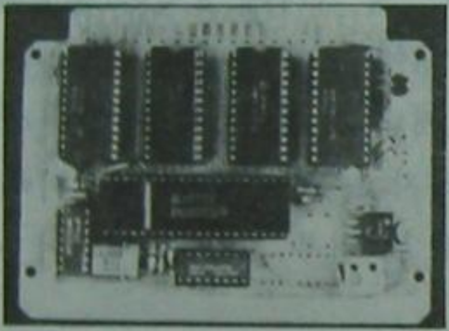


그림 2 기판에 IC를 꽂은 모습

이 같은 여러 개의 집적회로들을 모아서 하나의 커다란 회로를 구성한 칩을 LSI라고 하며 이 집적회로는 나름대로 하나의 시스템을 구성하고 있다. LSI의 예로는 Apple의 CPU인 6502 마이크로프로세서를 들 수 있으며 LSI보다 훨씬 집적도가 높은 칩을 VLSI라고 부른다.

③ **메모리 용량** : 메모리 용량은 크게 두 종류로 말할 수 있는데, 하나는 컴퓨터 내부에 있는 메모리의 용량이고 또 하나는 외부의 보조 기억 장치의 용량이다.

컴퓨터 내부의 메모리는 RAM과 ROM으로서 ROM은 컴퓨터를 동작시키는 시스템 소프트웨어나 BASIC 번역 프로그램 등의 컴퓨터 자체를 움직이는데 필요한 프로그램을 담고 있다. 일반적으로 ROM의 용량이 크면 더욱더 다양한 기능을 갖춘 컴퓨터가 된다.

RAM은 사용자의 프로그램과 데이터가 들어가는 메모리로서 일반적인 메모리 용량은 이 RAM의 용량을 말한다. RAM의 용량은 보통 K byte로 표시하는데, K는 Kilo의 약자로서 컴퓨터에서는 1000이 아닌 $2^{10}=1024$를 나타내며, Byte는 8 bit로서, 한 개의 문자나 숫자 데이터를 담을 수 있는 용량이다.

RAM의 용량이 클수록 컴퓨터는 큰 프로그램을 받아들일 수 있다.

④ **인터프리터** : 인터프리터는 고급언어인 BASIC 등을 컴퓨터가 알아들을 수 있는 기계어로 바꾸어주는 일종의 번역 프로그램이다. Apple의 6개 ROM 중 5개는 BASIC 인터프리터가 들어있어서, 사용자가 프로그램한 것들을 컴퓨터에게 번역, 전달해 준다. 인터프리터는 한 개의 프로그램 문장을 번역한 후 그것을 수

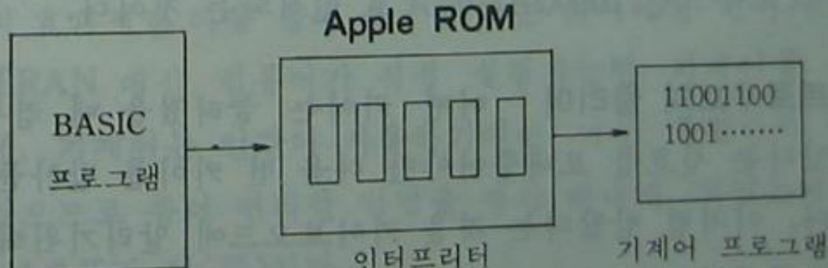


그림 3

행하게 하고, 또 한 개의 프로그램 문장을 다시 번역하게 되므로 프로그램 중간에 에러가 있더라도 바로 그 전까지는 수행해 가게 된다.

⑤ **모니터 ROM :** Apple의 6개 ROM 중 하나는 모니터 ROM이라 하여, Apple의 시스템 소프트웨어를 담고있는 비휘발성 메모리이다. 처음 Apple의 전원을 켤 때 '삑' 소리가 나고 화면 위에 Apple][라고 찍히는 것도 모니터 ROM의 작용이고, 키이를 누르면 화면에 글자가 찍히는 것도 모니터 ROM의 작용이다. 그러나 디스크를 작동시키는 프로그램은 이 안에 들어있지 않으며 그것은 따로 DOS라는 것에 만들어 두었다.

⑥ **아스키(ASCII)코우드 :** ASCII는 American Standard Code for Information Interchange의 약자로서 대부분의 마이크로컴퓨터는 ASCII 코우드를 쓴다.

이것은 7비트로 되어있으므로 총 $2^7=128$자까지를 정의할 수 있다. 이처럼 코우드를 사용하는 이유는 컴퓨터 내에서는 모든 문자나 숫자가 2진수의 형태로 돌아다녀야 하기 때문이다. 예를 들어 'A'라는 글자는 그 글자 자체가 컴퓨터에 인식되는 것이 아니고 해당 코우드인 1000001_2이 A로 인식되는 것이다.

⑦ **스트로우브의 클리어 :** 어떤 키이는 눌려졌을 때 컴퓨터가 이를 받았다는 신호를 보내주어야만 다음 번 키이를 받아들일 수 있게 된다. 이처럼 받았다는 것을 키이보오드에 알리기위해서 스트로우브 클리어 신호를 보내주게 되는 것이다. 보통 사용할 때는 모니터 프로그램에 의해 자동으로 처리된다.

⑧ **스크린 소프트 스위치 :** Apple에서 쓰이는 여러가지 화면의 선택(TEXT와 그래픽, GR과 고해상, Page 1과 Page 2, Full과 Mixed 등)을 하기위해서 POKE명령을 내려줄 수 있는데 이러한 소프트웨어 명령이 하드웨어적인 스위치 구실을 하여 화면의 모우드를 바꾸어 주므로 이것을 스크린 소프트 스위치라고 부른다.

예를 들어 HGR명령에는 다음 4개의 스위치 선택이 일어난다.

a. 그래픽 모우드

b. 고해상

c. Page 1

d. Mixed(혼합)모우드

⑨ **파라미터 :** 프로그램에서 파라미터란 한 프로그램 모듈에서 또다른 프로그램 모듈(예 : 서브루우틴)로 어떤 값을 전달하기 위해서 사용되는 매개변수(parameter)를 말한다.

또는 이 변수 자체가 프로그램의 데이터를 제공하는 수값을 갖기도 한다.

⑩ **어셈블리 어 :** 하드웨어의 콘트롤이나, 고속처리의 필요성, 메모리의 효과적인 이용 등이 요구될 때는 하이레벨 언어의 BASIC, FORTRAN 대신 컴퓨터가 직접 실행가능한 기계어를 사용하게 되는데, 기계어는 인간이 사용하기에는 너무 불편하게 숫자로만 되어있으므로 좀더 편리한 방법을 생각 해내게 되었는데 이것이 바로 어셈블리 어(語)이다.

어셈블리 어는 기계어 코우드에 짧은 명령어를 붙이고, 실제 주소대신 영문자로 레이블을 붙일 수 있게 되어 프로그래밍이 용

이하도록 하였다. 어셈블리 어 프로그램은 어셈블러라는 번역 프로그램에 의해, 그대로 기계어로 바뀌게 된다. 사용자는 이 바뀌어진 기계어 프로그램을 이용하는 것이다.

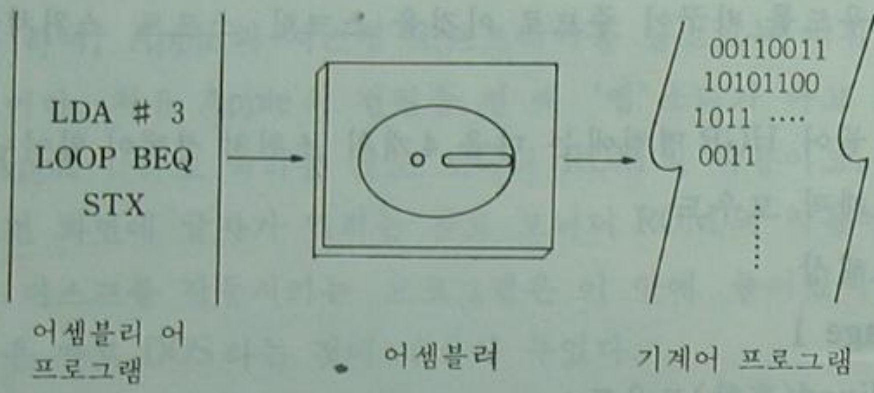


그림 4 어셈블러의 역할

⑪ **ROM Writer(롬 라이터)** : 이 장치는 ROM 정보를 써 넣을 수 있게 하는 장치로서 프로그램에 의해 조종되는 두 개의 전압에 의해, 들어갈 정보가 0이냐 1이냐를 결정하게 된다. 롬 라이터로 프로그램이 가능한 ROM을 PROM이라고 한다.

롬 라이터는, 컴퓨터에 부착시켜서 메모리에 있는 내용을 그대로 옮겨 주는 것과, 독립적인 키이를 사용하여 ROM에 기록시키는 것이 있다.

⑫ **EPROM** : 이 ROM은 ROM Writer에 의해서 정보를 써 넣을 수도 있고, ROM Eraser라는 자외선 발생기에 넣어두면 써진 정보를 다시 지울 수도 있는 ROM이다. 따라서 완성된 ROM을 만들기 전까지는 EPROM을 사용하여 몇 번씩 지웠다 썼다 하며 시행착오를 거치기 마련이다. 시중에서 구하기 쉬운 것으로는 2716

(2 K), 2732 (4 K)가 있다.

⑬ **Shape Table(모양표)** : 이 Table은 Apple soft BASIC의 DRAW, XDRAW 명령에 의해 HGR 화면 상에 어떤 모양을 나타낼 때, 그 모양을 정의해 주는 자료를 갖고 있는 표이다.

이 표는 한 개의 점이 찍히느냐 안찍히느냐, 다음 점으로의 이동 방향은 상, 하, 좌, 우 어느 쪽인가 등의 연속된 자료를 비트(bit)의 형식으로 가지고 있다.

⑭ **이니셜라이즈(INIT)** : 맨 처음 빈 디스켓을 사면 이곳에는 아무 것도 기록되어 있지 않다, 그러나 즉시 이 디스켓에 무엇을 써 넣을 수는 없는 것이다. 디스켓은 빈 노우트와는 달라서 일정한 규격을 잡아주는 작업을 해야 하는데, 말하자면 백지 노우트에 가로 줄을 그어서 글씨를 가지런히 쓸 수 있도록 해주는 것과 같은 것이다.

이 작업을 디스켓의 이니셜라이즈(initialize)라고 하는데 DOS가 메모리에 올라와 있는 상태에서

INIT 화일이름

의 명령을 내리면 빈 디스켓에는 DOS가 복사되며 일정한 틀이 잡혀서 사용가능한 상태가 되는 것이다. 이 때 이 화일을 그리팅 프로그램이라 부르며 디스켓을 넣고 컴퓨터의 전원을 넣었을 때 자동으로 맨 처음에 수행되는 프로그램이다.

부록 B

Apple의 메모리 구성

위치		memory type	용도
십진	Hex		
0~255	0~0F	RAM	시스템 프로그램
256~511	100~1FF	RAM	시스템 stack
512~767	200~2FF	RAM	키이보오드 input 버퍼
768~1023	300~3FF	RAM	Monitor vector locations
1024~2047	400~7FF	RAM	Text와 GR Page 1
2048~3071	800~BFF	RAM	Text와 GR Page 2
3072~8191	C00~1FFF	RAM	—자유 프로그램 부분—
8192~16383	2000~3FFF	RAM	HGR (Page 1)
16384~24575	4000~5FFF	RAM	HGR 2 (Page 2)
24576~49151	6000~BFFF	RAM	—자유 프로그램 부분—
49152~49279	C000~C07F	I/O	특수 내장 메모리
49280~49407	C080~C0FF	I/O	주변 장치 카드 I/O
49408~51199	C100~C7FF	I/O	주변 장치 카드 메모리
51200~53247	C800~CFFF	I/O	주변장치 카드 확장 메모리
53248~66535	D000~FFFF	RAM	APPLESOFT Autostart Monitor

부록 C ASCII 스크린 문자들

ASCII 스크린 문자들

	Inverse				Flashing				(Control)		Normal				(Lowercase)	
Decimal	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
Hex	$00	$10	'$20	$30	$40	$50	$60	$70	$80	$90	$A0	$B0	$C0	$D0	$E0	$F0
0 $0	@	P		0	@	P		0	@	P		0	@	P		0
1 $1	A	Q	!	1	A	Q	!	1	A	Q	!	1	A	Q	!	1
2 $2	B	R	"	2	B	R	"	2	B	R	"	2	B	R	"	2
3 $3	C	S	#	3	C	S	#	3	C	S	#	3	C	S	#	3
4 $4	D	T	$	4	D	T	$	4	D	T	$	4	D	T	$	4
5 $5	E	U	%	5	E	U	%	5	E	U	%	5	E	U	%	5
6 $6	F	V	&	6	F	V	&	6	F	V	&	6	F	V	&	6
7 $7	G	W	'	7	G	W	'	7	G	W	'	7	G	W	'	7
8 $8	H	X	(	8	H	X	(	8	H	X	(	8	H	X	(	8
9 $9	I	Y	)	9	I	Y	)	9	I	Y	)	9	I	Y	)	9
10 $A	J	Z	*	:	J	Z	*	:	J	Z	*	:	J	Z	*	:
11 $B	K	[	+	;	K	[	+	;	K	[	+	;	K	[	+	;
12 $C	L	\	,	<	L	\	,	<	L	\	,	<	L	\	,	<
13 $D	M	]	—	=	M	]	—	=	M	[	—	=	M	]	—	=
14 $E	N	^	.	>	N	^	.	>	N	^	.	>	N	^	.	>
15 $F	O	—	/	?	O	—	/	?	O	—	/	?	O	—	/	?

부록 D 키이와 해당 ASCII 코우드

Key	Alone	CTRL	SHIFT	Both
space	$A0	$A0	$A0	$A0
0	$B0	$B0	$B0	$B0
1!	$B1	$B1	$A1	$A1
2"	$B2	$B2	$A2	$A2
3#	$B3	$B3	$A3	$A3
4$	$B4	$B4	$A4	$A4
5%	$B5	$B5	$A5	$A5
6&	$B6	$B6	$A6	$A6
7'	$B7	$B7	$A7	$A7
8(	$B8	$B8	$A8	$A8
9)	$B9	$B9	$A9	$A9
:*	$BA	$BA	$AA	$AA
;+	$BB	$BB	$AB	$AB
,<	$AC	$AC	$BC	$BC
−=	$AD	$AD	$BD	$BD
.>	$AE	$AE	$BE	$BE
/?	$AF	$AF	$BF	$BF
A	$C1	$81	$C1	$81
B	$C2	$82	$C2	$82
C	$C3	$83	$C3	$83
D	$C4	$84	$C4	$84
E	$C5	$85	$C5	$85
F	$C6	$86	$C6	$86
RETURN	$8D	$8D	$8D	$8D
G	$C7	$87	$C7	$87
H	$C8	$88	$C8	$88
I	$C9	$89	$C9	$89
J	$CA	$8A	$CA	$8A
K	$CB	$8B	$CB	$8B
L	$CC	$8C	$CC	$8C
M	$CD	$8D	$DD	$9D
N^	$CE	$8E	$DE	$9E
O	$CF	$8F	$CF	$8F
p@	$D0	$90	$C0	$80
Q	$D1	$91	$D1	$91
R	$D2	$92	$D2	$92
S	$D3	$93	$D3	$93
T	$D4	$94	$D4	$94
U	$D5	$95	$D5	$95
V	$D6	$96	$D6	$96
W	$D7	$97	$D7	$97
X	$D8	$98	$D8	$98
Y	$D9	$99	$D9	$99
Z	$DA	$9A	$DA	$9A
→	$88	$88	$88	$88
←	$95	$95	$95	$95
ESC	$3B	$9B	$9B	$9B

찾아 보기

저 자 : 한 수 찬
서울대학교 공과대학 졸업
전 타이탄컴퓨터 개발실장
현 미국 유학 중

저 서 : Apple 입문(기전)
Apple 입문 II +(교학사)
Apple 어셈블리(교학사)
Apple PEEK, POKE 응용(교학사)
삼성 SPC-1000 연습(교학사)

편저서 : Apple 워드프로세서(과학정보사)
Apple 어셈블리 연습(교학사)

APPLE PEEK, POKE 응용

1984년 7월 10일 초판 발행
1987년 3월 20일 6판 발행

저 자 한 수 찬
발행자 양 철 우

발행처 (株) 敎 學 社

시 구로구 가리봉동 319-7
서울사무소 : 서울특별시 마포구 공덕동 105-67
전 화 : 영 업 717-4561~5
편 집 717-3557
대 체 : 012245-31-0501320
등 록 : 1962. 6. 26 〈7-13〉

값 2,500 원